Implementing Video Compression Algorithms on Reconfigurable Devices

Graeme Robert Stewart

MEng

A thesis submitted in fulfillment

of the requirements of the University of Glasgow

for the degree of Doctor of Engineering

Institute for System Level Integration

Faculty of Engineering

University of Glasgow

June 22, 2009

Abstract

The increasing density offered by Field Programmable Gate Arrays(FPGA), coupled with their short design cycle, has made them a popular choice for implementing a wide range of algorithms and complete systems. In this thesis the implementation of video compression algorithms on FPGAs is studied. Two areas are specifically focused on; the integration of a video encoder into a complete system and the power consumption of FPGA based video encoders.

Two FPGA based video compression systems are described, one which targets surveillance applications and one which targets video conferencing applications. The FPGA video surveillance system makes use of a novel memory format to improve the efficiency with which input video sequences can be loaded over the system bus.

The power consumption of a FPGA video encoder is analyzed. The results indicating that the motion estimation encoder stage requires the most power consumption. An algorithm, which reuses the intra prediction results generated during the encoding process, is then proposed to reduce the power consumed on an FPGA video encoder's external memory bus. Finally, the power reduction algorithm is implemented within an FPGA video encoder. Results are given showing that, in addition to reducing power on the external memory bus, the algorithm also reduces power in the motion estimation stage of a FPGA based video encoder.

Contents

List of Figures

List of Tables

Acknowledgments

This research was funded by the Engineering and Physical Sciences Research Council and 4i2i Communications Ltd. I am grateful to both for their financial support.

I am indebted to my academic supervisors, David Renshaw and John Hannah, for their advice and support, and for taking the time to read the many documents produced during the research period. Lastly, I thank them for encouraging me to submit my thesis in a timely fashion.

I thank my industrial supervisor, Martyn Riley, for giving me the opportunity to conduct research at 4i2i Communications. I am also grateful to my other colleagues at 4i2i, Duncan, Rob, Pamela, Mahdi, Murali, Steve, Willis, Andrew and Stuart, for their assistance and for making 4i2i Communications such an enjoyable place to work.

I thank everyone at the Institute for System Level Integration, for allowing me to study for an engineering doctorate and for making the journey such a rewarding experience.

Last, but by no means least, I would like to thank Clare, for the faith she has shown in me in the last 4 years and for putting up with my occasional grumpiness.

Graeme Stewart, August 2008

Author's Declaration

I declare that this thesis is my own account of my own research. Except for the work described in section 4.2, all the research described in this thesis was conducted by myself, with the assistance of my two academic supervisors David Renshaw and John Hannah. The research described in section 4.2 was conducted in conjunction with Rob Beattie, Chief Codec Architect at 4i2i Communications Ltd.

List of Abbreviations

ASIC ..... Application Specific Integrated Circuit

CIF ..... Common Intermediate Format (352x288 pixels)

DBM ..... Difference Based Mapping

DCT ..... Discrete Cosine Transform

DDR ..... Double Data Rate

DPCM ..... Differential Pulse Coded Modulation

EJO ..... Early Jump Out

FIR ..... Finite Impulse Response

FPGA ..... Field Programmable Gate Array

FSBMA ..... Full Search Block Matching Algorithm

GEA ..... Global Elimination Algorithm

IP ..... Internet Protocol

KLT ..... Karhunen Loeve Transform

LUT ..... Look Up Table

MPEG ..... Motion Picture Experts Group

OPB ..... Onchip Peripheral Bus

PSNR ..... Peak Signal to Noise Ratio

QCIF ..... Quarter Common Intermediate Format (176x144 pixels)

RAM ..... Random Access Memory

RDO ..... Rate Distortion Optimised

RTP ..... Real Time Transport Protocol

SAD ..... Sum of Absolute Differences

SATD ..... Sum of Absolute Transformed Differences

SDRAM ..... Synchronous Dynamic Random Access Memory

SEA ..... Successive Elimination Algorithm

SIF ..... Source Input Format (352x240 pixels)

SRAM ..... Static Random Access Memory

SSD ..... Sum of Squared Differences

VBM ..... Value Based Mapping

VBSM ..... Variable Block Size Matching

VCD ..... Value Change Dump

VGA ..... Video Graphics Adapter (640x480 pixels)

Chapter 1

Introduction

Advances in video compression have and continue to play a crucial role in the development of many communications applications. Applications such as digital TV, digital cinema, internet video streaming, video surveillance and video conferencing have all benefited from the improved quality and lower bit rates provided by the advancement in video compression techniques. The increasing processing capacity available to implement the video encoding and decoding functions has been crucial to realising this benefit in practice. Fundamentally, this is a result of the increasing density of transistors which can be placed on a silicon device.

The increasing transistor density has also caused the use of Field Programmable Gate Arrays (FPGAs) to change. When first introduced FPGAs were primarily used for glue logic because they had limited processing capacity. As their processing capacity has increased, the implementation of complete digital systems on FPGAs has become common. In this thesis, the implementation of video encoding systems on modern FPGAs is studied. Emphasis is on the implementation of the H.264 standard [1].

1.1 Motivation for Work

The characteristics which can be use to judge a video encoding implementation include,

- Supported frame rate and video resolution
- Supported Input Video Standards
- Video Quality for a given bitrate
- Power Consumption
- Latency
- Ease of Image Access
- Implementation Cost

The importance of each characteristic is dependent on the video encoding application. In digital television distribution applications the frame rate, video resolution and output video quality are key, with other factors being of secondary importance. Video conferencing applications also place a high level of importance on video quality, frame rate and video resolution. However, in this application these needs must be balanced against the need for both a low latency and comparatively lower cost encoding solution. For video surveillance applications video quality, frame rate and resolution may be sacrificed in order to reduce the unit cost and/or power consumption.

For any application an ASIC encoding solution offers the highest performance achievable in terms of both encoding performance, power consumption and unit cost. However, the cost of designing an ASIC is prohibitive. The design cost of

an ASIC can be in the tens of millions and is forecast to continue to increase [2]. In addition the long design cycle, and fixed nature of an ASIC can make it difficult to adapt to changes in the market and technology. For example the H.264 video compression system described in section 4.2 was an upgrade to a pre-existing video conferencing system which only supported older video compression standards. It is unlikely such an upgrade path would have been feasible if a pure ASIC solution had been used initially. As a result of the large design cost and inflexibility associated with an ASIC, reconfigurable solutions to implement the video encoding function are become increasingly attractive.

The reconfigurability can be provided, through general purpose processors or digital signal processors (DSPs), with the encoding functionality being defined in software, or, through the use of reconfigurable hardware. In some cases both reconfigurable hardware and digital signal processors are used to implement a complete encoding system [3]. If pursuing a software based video encoding implementation, DSPs are, in general, a better choice because they have performance, cost and power advantages over general purpose processors.

There are a wide range of DSP architectures available. To support H.264 encoding a high performance DSP architecture is required, particularly if high definition frame resolutions must be supported. Examples of such architectures include, the Stream Processor [4], which supports resolutions upto 1080p and a frame rate of 30 frame per second and the C64x architecture developed by Texas Instruments. This architecture supports the execution of up-to 8 instructions per second. Despite this high performance Texas Instruments have incorporated specific hardware, alongside the C64x DSP processor, to support high definition video encoding [5] in a number of their system on a chip devices.

One of the factors limiting DSP performance is the limit on the amount of data which can be transferred into the DSP subsystem from external memory [6] [7]. The stream architecture [4] has an innovative method to reduce this problem. The programming model for the stream architecture forces the designer to express the locality and concurrency associated with an application. The stream architecture's memory hierarchy is designed to exploit the expressed locality and concurrency. Consequently the external memory bandwidth requirements are reduced and the arithmetic logic units present in the stream architecture are able to perform useful calculations for a greater percentage of the time.

Despite the advances in the design of digital signal processors the amount of parallelism that can be exploited is still limited. For example the stream processor described in [4] can only exploit data level parallelism. In the case of video encoding this forces it to operate on multiple image blocks at once to achieve the desired throughput [4]. This is, arguably, not an ideal way to implement a video encoding solution because, as will be discussed in chapter 2, a number of the algorithms which can be used in a video encoder rely on knowledge of how adjacent blocks have been encoded. In contrast reconfigurable hardware allows the type and level of parallelism to be exploited to be determined by the needs of the application. This allows a high level of performance to be achieved.

FPGAs are currently the most commercially successful reconfigurable hardware platform available today. They contain fine grained logic elements which accept 3-6 single bit inputs, a range of resource to support arithmetic operations, and a range of memory resources. This allows them to implement both the functional blocks and the memory hierarchy required by a video encoding solution. While FPGAs can realise the high level of performance on offer by using reconfigurable logic, they do have limitations. There is substantial area and power

overhead associated with using an FPGA. This is principally a consequence of the large degree of reconfigurability they support.

Coarse grained logic arrays, where the basic logic elements are much larger and more complex than in a conventional FPGA, are another form of reconfigurable hardware which could be used to implement a video encoder, potentially with more area efficiency and lower power than that offered by a conventional FPGA. However, currently these devices are not of sufficient maturity. This is especially true in the case of the development tools required. However, given that they have some similarity to conventional FPGAs some of the research presented in this thesis may also be applicable to such devices.

In summary, reconfigurable logic offers the highest performance means of implementing a video encoder whilst maintaining a degree of flexibility. FPGAs are the most commercially viable reconfigurable logic devices available currently. It is for these reasons that FPGAs are focused on in this thesis. Two areas are specifically focused on,

- the integration of a video encoder into a complete FPGA based system
- the power consumption of FPGA video encoder implementations.

Video compression systems are studied because the integration of a video encoder into a complete system has a direct impact on some of the characteristics listed. Specifically the latency of, and the resources used by, the encoding solution are significantly affected by the overall system design. The system design intellectual property and associated tools offered by the two main FPGA manufacturers, Altera and Xilinx, are used to realise two distinct video compression systems. Given the ready availability of this intellectual property, the insights gained in each case will be applicable to a range of video encoder system designs.

As increasingly complex systems have been implemented on FPGAs, power consumption has become an increasing concern. Consequently, a significant amount of research has been conducted into methods used to reduce it, as discussed in chapter 3. As previously mentioned the importance of power consumption varies depending on the video encoding application. However, in all applications it is desirable to reduce FPGA power consumption. Excessive FPGA power consumption will increase heat sink, packaging and operating costs and lower device reliability.

The H.264 standard offers substantially reduced bit-rates for equivalent video quality compared to that offered by previous compression standards [8]. A number of new encoding tools, including variable block sized motion estimation, intra prediction and multiple reference frame prediction, provide this bit-rate improvement. When each encoding tool is used encoding complexity increases [9], and therefore, the power consumed by the encoder increases. The difficulty in achieving an acceptable level of power consumption is further complicated both by the desire to support high-definition resolutions in many video encoding applications and the power overhead associated with FPGA reconfigurability. Thus, the central aim of the majority of research described in this thesis was to highlight techniques to reduce power consumption in FPGAs and apply them to FPGA video encoding systems

1.2 Contribution Of Thesis

The significant contributions of this research to FPGA video encoder implementation field and more generally to the video encoder implementation field are.

- A novel memory format which allows the input frames required by a video

encoder to be transferred to and from an external memory more efficiently.

- An analysis of the power consumed by an FPGA based video encoder. A number of published papers have described complete FPGA video encoder implementations [10] [11] [12] [13] [14] [15]. However, none have characterised the power consumption as is done in this thesis.

- The development of an algorithm to reduce the power used loading and saving reference frames from external memory. The algorithm developed is unique in that it can be used in conjunction with other more generic bus encoding algorithms, such as the bus invert algorithm [16].

- The implementation of the proposed algorithm into a FPGA based pipelined video encoder. Results are provided showing that the algorithm also reduces power in the motion estimation stages of a FPGA based video encoder.

- A novel low power FPGA architecture for the Full Fractional Search Algorithm. In contrast to other proposed Full Fractional Search implementations which either repeat the half pixel interpolation operation [17] [18], or store the half pixel samples in registers [19], the proposed architecture stores the half pixel samples in FPGA embedded RAM. Results are provided showing the power benefit of the approach taken compared to when the half pixel interpolation is repeated.

All of the proposed techniques can be viewed as FPGA implementation variations of standard video encoding processes. None of the proposed techniques sacrifice encoding performance in order to achieve a reduction in the power consumption or resources required by a FPGA video encoder. The above novel contributions have led to the following publications,

- G. Stewart, D. Renshaw, and M. Riley, "A low-cost, fpga based, video streaming server," in *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, 2007, pp. 187–190. [Online]. Available: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4234343`

- G. Stewart, D. Renshaw, and M. Riley, "A novel motion estimation power reduction technique," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 546–549. [Online]. Available: `http://dx.doi.org/10.1109/FPL.2007.4380713`

- G. Stewart, D. Renshaw, and M. Riley, "Power savings in fpga video compression systems through intra prediction result reuse," in *Programmable Logic, 2008 4th Southern Conference on*, 2008, pp. 161–166. [Online]. Available: `http://dx.doi.org/10.1109/SPL.2008.4547749`

1.3 Thesis Organisation

The rest of this thesis is organised as follows,

- **Chapter 2** describes the algorithms and architectures used for video encoding, with an emphasis on the H.264 video compression standard. The hybrid DPCM/DCT video encoding framework is introduced. This is followed by a discussion of the various algorithms and architectures used to implement each section of it.

- **Chapter 3** describes the basic architecture used by the majority of FPGAs available today. The sources of power consumption in an FPGA are identified and the general techniques used to reduce FPGA power consumption discussed.

- **Chapter 4** focuses on FPGA video compression systems. Two video compression systems are described; one for video surveillance applications; one for video conferencing applications. The novel input image memory format is proposed in this chapter.

- **Chapter 5** analyses the power consumption of a pipelined H.264 video encoder when implemented in an FPGA.

- **Chapter 6** proposes an algorithm to reduce the power used by an FPGA video encoder's memory bus and provides results showing its effectiveness. In addition, results are provided showing that the proposes algorithm is compatible with other more generic bus encoding algorithms.

- **Chapter 7** discusses the implementation of the algorithm proposed in chapter 6 in a pipelined video encoder. An implementation of the full search and full fractional search motion estimation algorithms is then described. Finally, results are given showing that the algorithm proposed in chapter 6 reduces the power consumed in the motion estimation stage of a FPGA video encoder in addition to reducing the power consumed on the external memory bus.

- **Chapter 8** - Concludes the thesis and provides suggestions for further work.

Chapter 2

Video Compression: Algorithms and Architectures

Video compression is an extensively studied subject. Both the algorithms used for it and the architectures used to implement it have been the subject of significant research. In modern video compression systems a variety of algorithms are employed. The purpose of each algorithm is either, to exploit the redundancy inherent in natural video sequences or to remove information from a video sequence which is of negligible benefit to the perceived video quality.

Despite the large amount of research which has been conducted into video compression the same hybrid DPCM/DCT [20] framework has been used in the majority of video compression standards. The increase in compression performance achievable using newer standards such as H.264 [1] is a result of improvements to the processes already used within the hybrid DPCM/DCT framework. In addition the H.264 standard supports two new processes, intra prediction and in-loop filtering, to further increase the achievable compression performance.

There are other standards, such as JPEG-2000 and Dirac, which are based on

wavelets. Currently these standards are mainly used in applications where the quality of the compressed video is a critical requirement, for example, the JPEG-2000 standard is used for digital cinema. In this thesis, the hybrid DPCM/DCT framework is focused on because it is the method used by the majority of video compression standards, and consequently the method used by the majority of video compression applications.

2.1 Hybrid Video Compression Overview

2.1.1 Digital Video

A digital video consists of a sequence of pictures captured at regular time intervals. Each picture consists of one, or more, rectangular arrays of quantised samples (pixels) containing intensity and colour information. Different formats can be used to represent the intensity and colour information. For video coding applications the YCrCb format is generally employed. In this format three rectangular arrays are used [21]. The Y or luma array contains the intensity information. The two chroma arrays, Cr and Cb, contain the colour information.

The human visual system is less sensitive to colour information than it is to intensity information. Therefore, it is common that the the size, or resolution, of the two chroma arrays is smaller than that of luma array. Figure 2.1 illustrates three of the chroma sub-sampling formats currently in use. In 4:4:4 sub-sampling, the resolution of the two chroma arrays is identical to that of the luma array. In 4:2:2 sub-sampling, the horizontal resolution of each chroma array is half that of the luma array. In 4:2:0 sub-sampling, both the horizontal and vertical resolution of each chroma array is half that of the luma array. Only video sequences with 4:2:0 sub-sampling are used in this work. Other sub-sampling patterns are

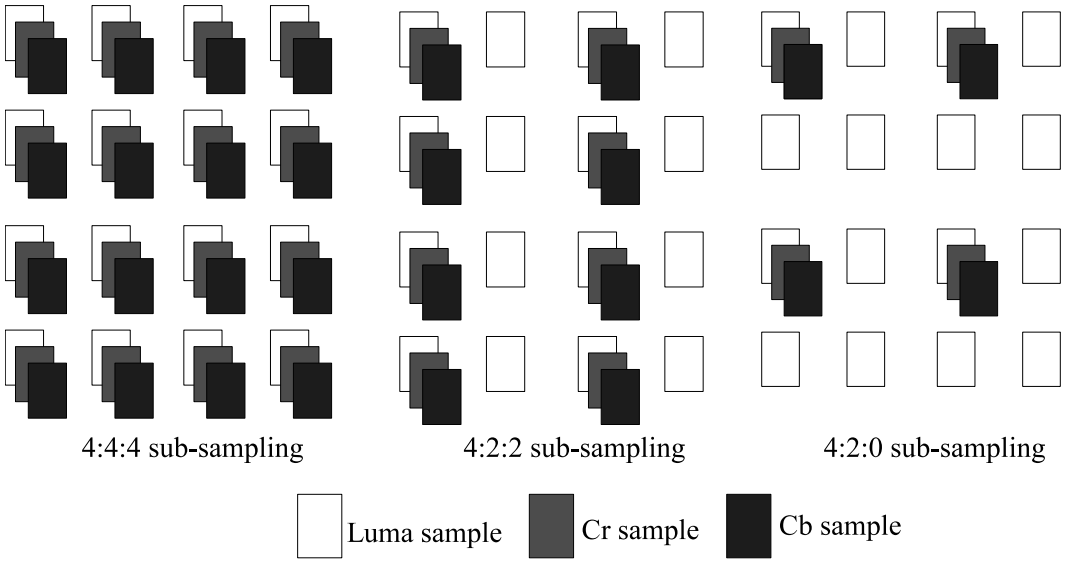

Figure 2.1: Different chroma sub-sampling formats currently in use

supported by the H.264 standard [1].

A picture can consist of either a single video frame or two video fields. All the samples in a video frame are captured at the same time instant (progressive video). Each video field contains either the odd or even picture lines, with each field being sampled at a separate time instant (interlaced video). The H.264 standard provides specific modes to support the encoding of interlaced video sequences. In this thesis, however, only progressive video is used.

The rate at which frames are captured, the frame rate, determines how smoothly motion is represented. Frame rates of 25 or 30 frames per second are typical. In some applications frame rates of up to 60 frames per second are used.

2.1.2 Video Encoding Process

Two types of redundancy in a video sequence can be readily identified,

- Spatial redundancy within an individual frame of the video sequence. Large

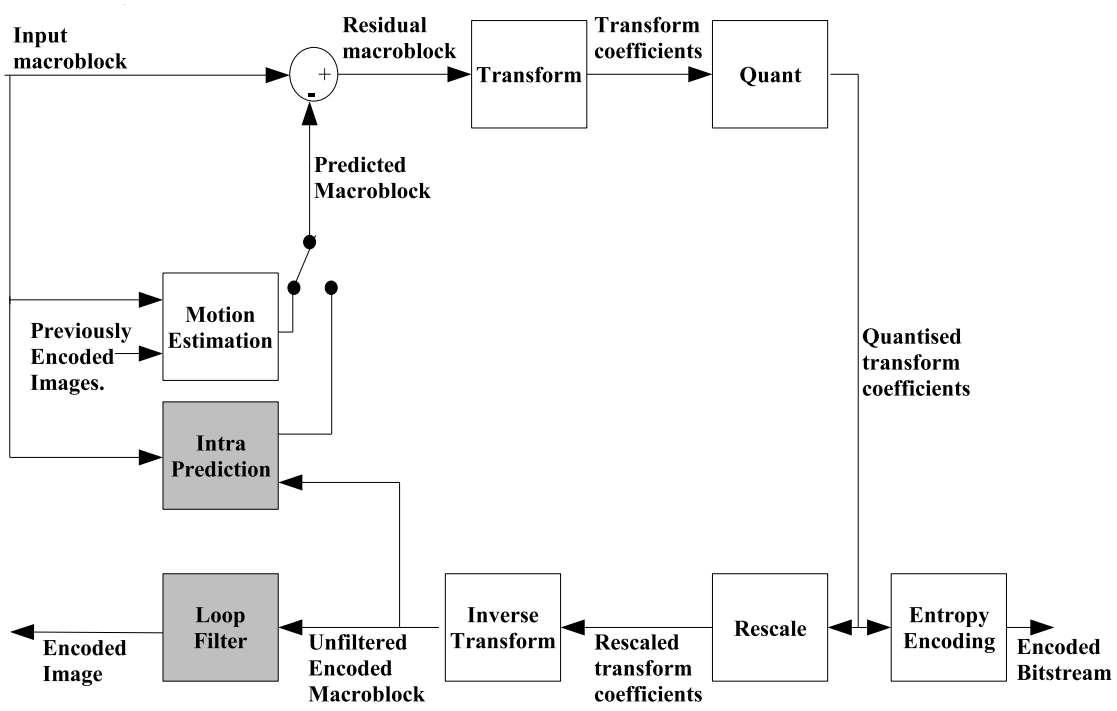

Figure 2.2: The hybrid encoding process. Each box represents a stage of the hybrid encoding process. The shaded boxes represent stages that are only used within the H.264 standard.

areas within a single frame may have similar luminance and chrominance characteristics.

- Temporal redundancy within a video sequence. Temporally adjacent frames within a video sequence are generally very similar. This is exploited to achieve much greater compression than would be achieved if spatial redundancy alone was considered during the compression process.

A diagram of the hybrid video compression process is shown in Figure 2.2. The hybrid video compression process is block based. Each image to be encoded is divided into non-overlapping 16 by 16 pixel blocks as illustrated in Figure 2.3. Each 16 by 16 pixel block, or macroblock, is processed separately through the majority

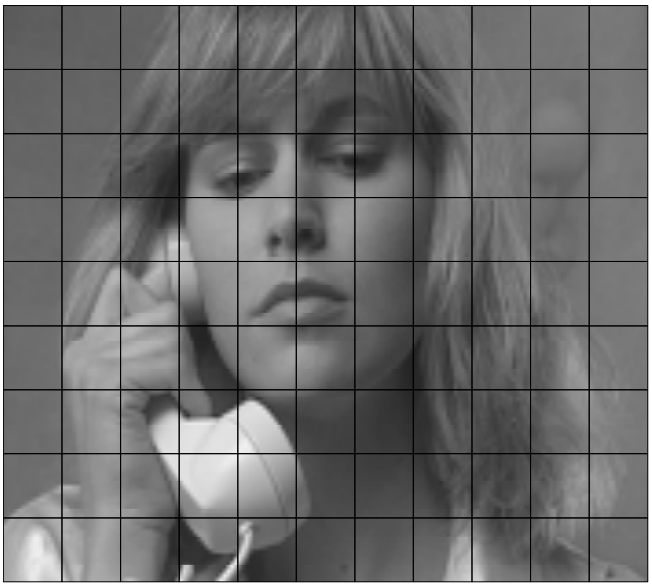

Figure 2.3: A QCIF image divided into 99 16 by 16 pixel blocks. Each 16 by 16 pixel block is a macroblock. Each macroblock is processed separately through the majority of the encoding stages as shown in Figure 2.2

.

of the encoding stages. This is advantageous because the temporal and spatial characteristics of different sections of each video frame are not identical. Therefore, by considering such sections separately better compression performance is obtained.

The first two stages of the encoding process are intra prediction and motion estimation. Both analyse the video sequence in order to aid the compression process. The motion estimation stage identifies and compensates for the motion of a macroblock between frames of a video sequence. This aids the exploitation of the temporal redundancy present in the video sequence. Similarly the intra prediction stage identifies the direction in which a macroblock, or part of it, has greatest spatial redundancy. This aids the exploitation of the spatial redundancy present in the video sequence.

A residual macroblock is formed from the difference between the input macroblock and the macroblock prediction made by either the motion estimation or intra prediction encoder stage. The transform stage de-correlates the residual macroblock producing output coefficients. Ideally the output coefficients will have the majority of their energy concentrated in as few a number of coefficients as possible [20]. Information is lost in the quantisation stage. The transform coefficients are rounded into a limited range of values, dependent on the quantisation parameter used. The quantisation parameter is the primary mechanism used to control the size of output bitstream and the quality of the encoded frames [20]. If a constant quantisation parameter is used the quantisation parameter is kept the same for every frame in the video sequence. This results in all the encoded video frames being of similar quality. However, the number of bits required to encode each frame, the bitrate, will vary depending on the complexity of the frame being encoded. In many applications this is undesirable. Thus, frequently the quantisation parameter is varied, on a frame or marcoblock basis, in order to control the output bitrate [20].

As a result of the information lost through quantisation each input macroblock cannot be perfectly reconstructed by the decoder. The quantised coefficients and the ancillary information associated with a macroblock (motion vector, macroblock mode etc) are then put through an entropy encoding process to produce the encoded bitstream.

The other stages within the hybrid video compression process, rescaling, inverse transformation and loop filtering are used in both an encoder and a decoder. In an encoder they are required to provide encoded data for the motion estimation and intra prediction encoder stages. An encoder uses encoded data for intra prediction and motion estimation to ensure that the data it uses matches that

available at the decoder.

The rescaling, inverse transform and reconstruction stages generate encoded macroblocks from the quantised transform coefficients and the predicted macroblock pixels. The loop filter attempts to remove any artifacts from the reconstructed frame which may have been introduced as a result of the encoding process. The loop filter does not operate solely on a macroblock basis. It must filter at the boundaries between macroblocks because often, it is at these boundaries that any artifacts are most apparent [20].

2.1.3 I,P and B Slices

A frame, or a segment of a frame (a slice), can be encoded as either an

- **I Slice** - in which each macroblock can be encoded using only the intra prediction modes supported by the encoder.
- **P Slice** - in which each macroblock can be encoded using either an intra prediction mode or a motion estimation mode where each block has at most one motion vector and reference index associated with it.
- **B Slice** - in which each macroblock can be encoded using either an inter prediction mode or a motion estimation mode where each block has at most two motion vectors and reference indices's associated with it.

P and B slices exploit the temporal redundancy present in a video sequence. As a result they can achieve a much greater level of compression that I Slices, which only exploit the spatial redundancy within a single frame of a video sequence. I Slices are used to provide an initial frame for motion estimation and macroblock prediction and to provide access points in the encoded bitstream.

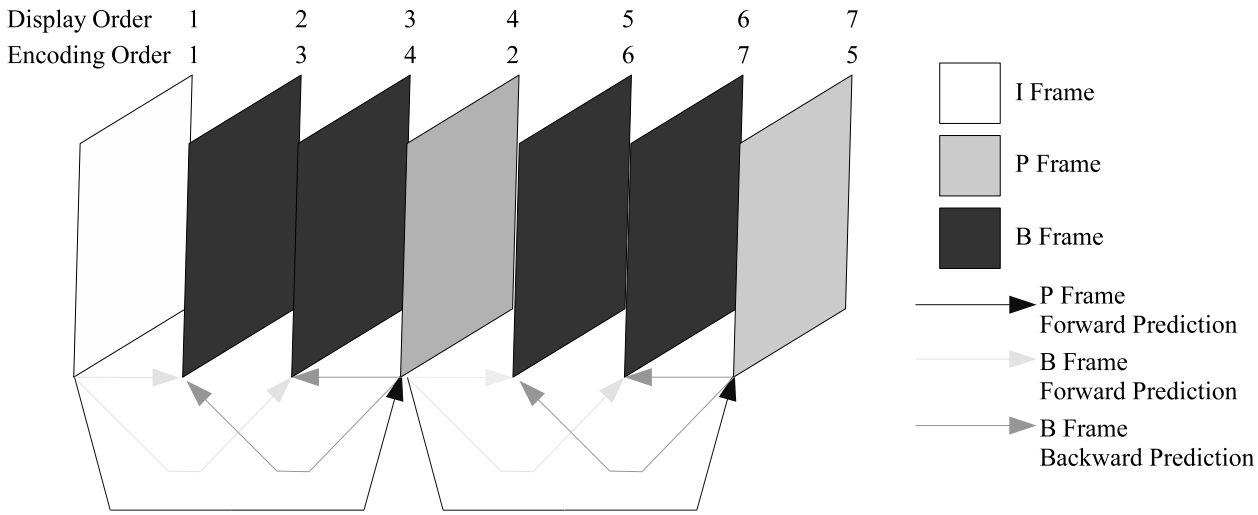

Figure 2.4: An encoded video sequence containing I,P and B frames

.

As illustrated in Figure 2.4 macroblocks in a P Slice are predicted from one reference frame. The reference frames used in a P slice are encoded images which will by displayed prior to the frame being encoded. B Slices allow each predicted block to be formed from an average of blocks present within two reference frames. In previous standards, the two reference frames used were fixed, one from the past and one from the future in display order. The H.264 standard allows the two reference frames used to be determined by the encoder. In the majority of instances, a forward and backward reference will still be used however, as shown in Figure 2.4. B slices are used because forming the predicted block from multiple reference blocks improves compression efficiency.

2.1.4 Measuring Video Quality

As a result of the quantisation stage, information is lost during the encoding process. Consequently, the quality of each encoded frame is less than that of

each unencoded frame. To quantify the quality loss the PSNR metric is typically used. It is given by,

$$PSNR(db) = 10\log_{10}(2^n - 1/MSE) \tag{2.1}$$

where n is the bit width of the quantised samples and the mean squared error (MSE) for a M by N sample array is given by

$$MSE = \frac{1}{M * N}\sum_{i=1}^{M}\sum_{j=1}^{N}(P_{encoded}(i,j) - P_{unencoded}(i,j))^2 \tag{2.2}$$

where $P_{encoded}(i,j)$ and $P_{unencoded}(i,j)$ represent the encoded and unencoded pixels at position (i,j).

2.1.5 Video Compression Standards

Two main standardisation bodies have produced video encoding standards,

- Video Coding Experts Group (VCEG) of International Telecommunications Union Telecommunication Standardization Sector (ITU-T)
- Motion Picture Experts Group (MPEG) of International Organization for Standardization International Electrotechnical Commission (ISO/IEC)

The Video Coding Experts Group produced the H.261 and H.263 standards. The Motion Picture Expert Group produced the MPEG-1, MPEG-2 and MPEG-4 standards. The H.264 standard was jointly developed by the two standardisation bodies. More details on each standard can be found in [22] and [20].

2.2 Video Compression System Architectures

The architectures proposed for FPGA video encoding include, dedicated hardware compression pipelines [11] [23] [14], mixed hardware/software based systems where specific tasks are performed in hardware accelerators [24] [25] but the less computationally intensive tasks are performed in software and pure software based encoders [12]. Some degree of hardware acceleration is generally required in an FPGA in order for the encoder to operate at a sufficient frame rate. A FPGA based MPEG-4 software encoder was proposed in [12]. Multiple Nios-2 processors were used to encode separate slices of each video frame. Even when using 3 Nios-2 processors for encoding the compression system proposed in [12] was only capable of encoding QCIF frames at a rate of 6 per second.

One of the justifications for including a software element in a compression system is that it provides a degree of flexibility [12] [26]. Given the flexibility inherent in FPGAs this justification has less weight than it would if the compression system was implemented using an ASIC technology. Therefore, in this thesis the focus is primarily on hardware encoder implementations. It is assumed that any processor present within the system is used solely for encoder control and scheduling operations.

A macroblock pipeline is the predominant architecture used in hardware implementations [11] [23] [14] [10] [27]. This is unsurprising given the block based, sequential nature of the encoding process. Pipelining allows the various encoder modules to operate simultaneously. This improves an encoder's resource utilisation and increases the throughput an encoder can achieve. However, the performance improvement comes at a cost. A significant number of embedded RAMs are required to support pipeline operation. In [15] a generic video processing ar-

chitecture is proposed. The analysis in [15] focuses on the design of the communications architecture between the processing elements within a pipeline, providing a methodology which can be used to ensure the processing pipeline as a whole operates at the rate required.

While the general architecture is the same, there are differences between the various macroblock pipeline implementations which have been proposed. The number of pipeline stages varies. H.264 implementations generally have a larger number of pipeline stages than implementations supporting previous standards [23] [27]. This can be attributed to the greater complexity of the H.264 encoding process. For example the motion estimation process in H.264 supports both variable block sizes and quarter pixel prediction. As a result, there is a greater benefit from splitting the motion estimation process across a number of pipeline stages than there would be if implementing standards with a less complex motion estimation process.

The method used to control the macroblock pipeline also varies between implementations. The simplest method, used in [23] [28], is to use a fixed number of clock cycles per pipeline stage, with each stage in the macroblock pipeline advancing to the next macroblock after a set number of clock cycles. In this case the maximum frame rate an encoder can operate at is given by,

$$F_r = F/(w_{mb}h_{mb}C_p + N_p) \tag{2.3}$$

where w_{mb} and h_{mb} are the frame width and frame height in macroblocks, F_r is the required frame rate, F is the clock frequency of the encoder, N_p is the number of pipeline stages and C_p is the number of clock cycles required before the pipeline can advance.

More flexible control schemes have been used [11] [14]. In [11] each individual stage in the pipeline advances to the next macroblock as soon as it completes operations on the current macroblock. This is conditional on the next macroblock being available for processing and the succeeding pipeline stage being able to accept data. In [14], the macroblock pipeline advances as soon as all stages have completed operations for their current macroblock. The benefit of a more flexible pipeline control scheme is that each macroblock can use a different number of clock cycles at each encoder stage. This can potentially allow a reduction in the clock frequency required to support a particular frame rate and resolution. Any actual benefit is, however, dependent on the algorithms and architectures used within the encoder, particularly those used for motion estimation and mode decision.

Apart from pipelining, other methods can be used to parallelise the encoding process. As previously mentioned each frame can be split into several slices, as shown in Figure 2.5. This allows a separate encoder instance to be used to encode each slice. Using this method of parallelisation, any redundancy that exists between adjacent macroblocks that are in separate slices cannot be exploited. As H.264 considers this redundancy, this method of parallelisation reduces the video quality that is achievable for a given bit rate [29]. Parallelisation can also take place at the frame level. In this case, each separate encoder instance is used to encode a subset of the pictures within the video sequence [30]. This method is only practical if B frames, which use a forward and a backward reference frame, are used within the encoded video sequence. As such this method of parallelisation is unsuitable in low latency applications.

Frame and slice based parallelisation are independent of the architecture used to implement each encoder instance. Using an efficient architecture such as a

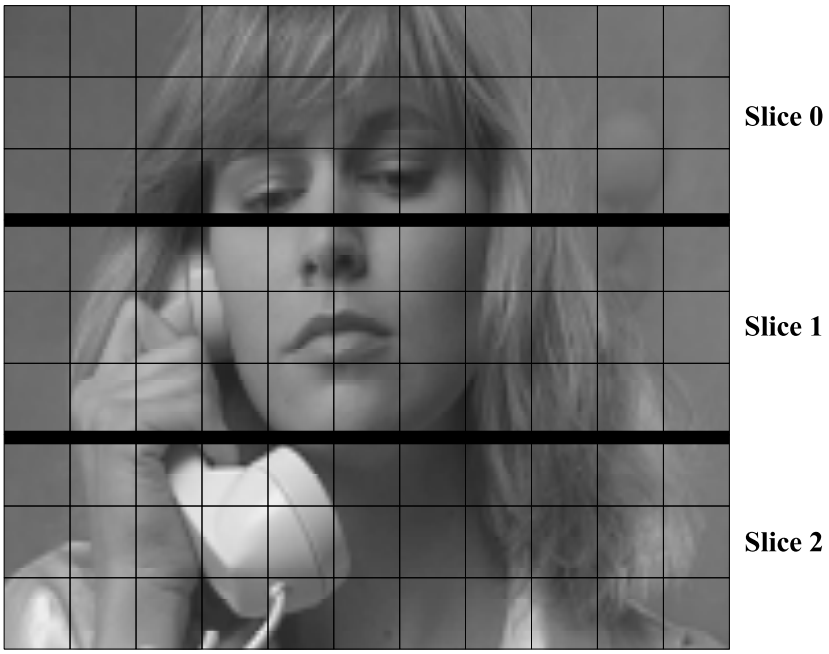

Figure 2.5: Splitting a video frame into several slices to enable parallelisation

macroblock pipeline reduces the need to use these other methods. The benefit of using slice or frame based parallelisation is that it allows the total hardware resources to scale easily with the required video frame rate and resolution.

2.3 Motion Estimation

In general, the aim of the motion estimation operation is to determine the displacement of objects between two frames in a video sequence. The methods used for motion estimation can be put into two categories, intensity based methods or feature/object based methods. Intensity based algorithms look for correspondence between images based on the luminance values within each image. Feature/object based methods first derive a set of features from each image then look for correspondence between the features present in each set. The majority of video compression standards imply that block based matching, an intensity

based method, should be used. Therefore, it is this method which is focused on in this thesis

The basic fixed sized block matching process suffers from a number of limitations. As a consequence the residual energy in the difference frame may not be minimised. This in turn impacts the compression performance of the encoding system. Either more bits are required to encode the residual frame, or at low bit rates visible blocking artifacts are introduced. Improvements to the basic block matching process have been one of the key reasons why video compression performance has improved with successive video compression standards [21]. These improvements include

- Fractional Pixel Motion Estimation
- Unrestricted Motion Vectors
- Variable Block Sizes
- Multiple Reference Pictures
- Bi-predictive Motion Estimation (B slices)

While these refinements to the basic matching process have increased compression performance they have also increased complexity. The addition of fractional pixel motion estimation, for example, typically requires that motion estimation operation to be a two stage process. An integer pixel motion estimation operations is followed by a separate fractional pixel refinement stage. A significant amount of research has been conducted into both reducing the complexity of the motion estimation process and implementing it efficiently in various types of hardware [20] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40].

2.3.1 Full Pixel Motion Estimation: Algorithms

The full pixel estimation problem can be defined as follows. The frame to be encoded is divided in into a number of non-overlapping equally sized blocks of size $N_h * N_w$ pixels. For each block in the image to be encoded, an area of the reference frame is searched in order to find a block of size $N_h * N_w$ which minimises a cost measure. The displacement between the current frame block and the candidate block with the minimum the cost is the motion vector to be refined during the fractional pixel estimation process. If variable block sizes are used, a motion vector must be found for each block size.

In H.264 the cost measure generally includes two terms. The first term is a distortion measure. This gives an indication of the level of similarity between the current frame and candidate blocks. The second term is an estimate of the number of bits required to encode the ancillary (motion vector used, reference frame used) information for the candidate block being considered. This rate term is designed to have more significance at higher quantisation levels because at higher quantisation levels, the ancillary information occupies a greater proportion of the overall bitrate [8]. Thus, to calculate the overall cost J_{motion} for a candidate block with motion vector (x, y) and reference frame r, the following equation is used

$$J_{motion}(x, y, r) = D(x, y, r) + \lambda_{motion} R(x, y, r) \tag{2.4}$$

where $D(x, y, r)$ is the distortion measure, $R(x, y, r)$ is the rate, and λ_{motion} is the bias applied to the rate term. When a rate term is inlcluded, the motion estimation problem is formulated as a rate/distortion problem. In principle λ_{motion}, for a given motion vector field rate $R_{motion-limit}$ could be found through a bisection search [41]. In general this is not practical. In the H.264 reference model λ_{motion}

is determined empirically, as a function of the quantisation parameter [8].

The simplest algorithm which can be used for motion estimation is the full search block matching algorithm (FSBMA). In this algorithm the cost is calculated at each position within the reference image search area. Thus, using the FSBMA the candidate block with minimal cost will always be found. The algorithm is computationally complex however. When encoding a CIF image at 30 frames per second approximately 9.3 giga-operations per second are required [42], assuming the Sum of Absolute Differences (SAD) distortion metric is used. Many algorithms have been proposed which attempt to reduce the computational complexity of the motion estimation process whilst providing compression performance similar to that of the FSBMA. The methods used include, reducing the number of search positions, using previously computed motion vectors for motion vector prediction, reducing the complexity of the distortion metric used, early elimination of candidate blocks and the utilisation of multi-resolution reference images.

Search Position Reduction

This method of complexity reduction focuses on devising search patterns which reduces the number of candidate blocks which have to be evaluated. The aim being to devise a search pattern which locates the optimum candidate block whilst requiring the minimum number of search positions to be tested. Examples of such algorithms include the Three Step Search, New Three Step Search [31], One Dimensional Search [32], Four Step Search [33], Diamond Search [34] and Hexagonal Search [35]. Algorithms such as the Three Step Search and one dimensional search are primarily based on the monotonicity assumption. This states that the block distortion measure will increase monotonically as the point being measured

moves away from the global optimum. Newer algorithms such as the Diamond Search [34] and Hexagonal Search [35] also incorporate the assumption that the optimum block will probably be located at, or close to, the search centre.

Distortion Measures

The distortion measure typically used for the motion estimation operation is either the Sum of Absolute Differences (SAD) or the Sum of Squared Differences (SSD). For motion vector **m** both the SAD and the SSD can be calculated using the following equation,

$$D = \Sigma_{x,y \epsilon A} |f_1(x, y) - f_2(x + m_x, y + m_y)|^p \tag{2.5}$$

where f_1 is the frame being encoded, f_2 is the reference frame, and p is 1 for SAD and 2 for SSD. A number of alternatives to these distortion measures have been proposed. The majority have aimed to reduce the complexity associated with the motion estimation process. There have been some distortion measures proposed however which aim to increase compression performance such as the Sum of Absolute Transformed Differences(SATD) [20].

Truncating the least significant bits of the reference and current pixels prior to evaluating equation 2.5 provides a simple method with which to reduce distortion metric complexity. If the number of bits truncated is fixed this method can reduce both the resources and power used by the motion estimator. If the number of bits truncated is varied [36], only the power can be reduced using this method. The reduction in power is typically greater than the proportion of bits truncated because the switching activity of video data tends to be concentrated in the least significant bits [37].

The advantage of varying the number of bits truncated is that the reduction in compression performance caused by bit truncation can be reduced when necessary. The impact bit truncation has on compression performance is dependent on the number of bits truncated and the sequence being encoded. For many sequences truncating the 4 least significant bits results in no significant reduction in compression performance when only a fixed block size of 16x16 is used [36]. In [37] it is shown that, for the smaller block sizes supported by the H.264 standard, bit truncation causes a greater degradation in motion vector accuracy. The effect on compression performance is less pronounced because , in general, smaller block sizes are only used to encode a small proportion of macroblocks in a frame. Results presented in [43] indicate that even when supporting smaller block sizes, truncating the least 3 significant bits has a negligible effect on H.264 compression performance.

Sub-sampling current and candidate blocks is another method which has been proposed to reduce distortion metric complexity [44] [45]. With this method, the distortion metric is calculated using a subset of the pixels in each block as indicated in Figure 2.6. As a result of the frequency aliasing introduced by sub-sampling, motion vector accuracy and hence compression performance can be reduced. In [44], the performance impact of sub-sampling is reduced by varying the sub-sample pattern used for each search position. In [45], edge detection techniques are used to determine the location of high frequency components in the current block. Sub-sampling is then only applied at locations where high frequency components were not detected. Similarly to bit truncation, using an adaptive sub-sampling pattern reduces the hardware resource savings which can be realised. In [46], instead of using an adaptive sub-sample pattern, a low pass filter is used to reduce the magnitude of the high frequency components prior

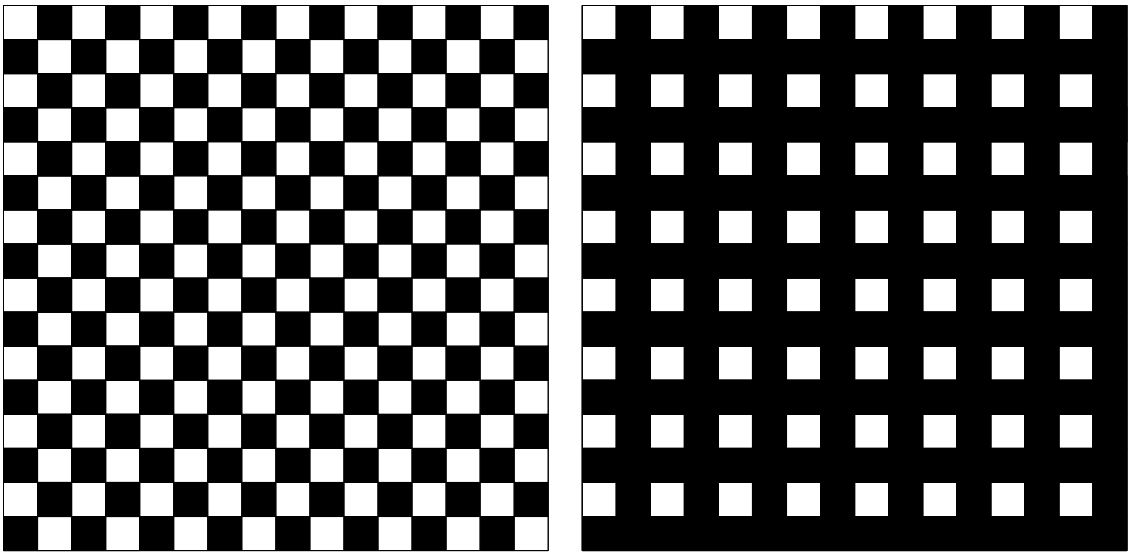

Figure 2.6: Example 2 to 1 (left) and 4 to 1 (right) sub-sampling patterns, only the unshaded pixels are used in the SAD calculation

to sub-sampling. This method allows the hardware resources required for the motion estimator to be reduced. However, it does require additional resources for the low pass filter implementation.

Both sub-sampling and bit truncation reduce complexity by simplifying the standard SSD/SAD distortion measure. Simpler distortion measures which are not based on the standard SSD/SAD measure have also been developed. Examples include,

- **mini-max** where the distortion measure D is the greatest absolute pixel difference between the reference and current block [47]
- **pel difference classification**(PDC) where each pixel is classified as a match or not. The distortion measure in this case is the total non-matched pixels.

The alternatives to the SAD and SSD have not been widely adopted because

of their relatively poor compression performance. Results provided in [48] indicate that the PDC measure offers better compression performance than the SAD measure. However, the performance of PDC measure is heavily dependent on the threshold used for classification and the sequence being compressed.

Full Search Complexity Reduction

Two algorithms, the Early Jump Out Algorithm (EJO) and the Successive Elimination Algorithm (SEA) [38], obtain the same result as the full search algorithm but with reduced complexity. The Early Jump Out Algorithm is simple. The SAD accumulation for a particular position is terminated if the partial SAD value is greater than the current best SAD. Due to its serial nature, the EJO algorithm is most applicable to software implementations. There have been hardware architectures proposed which implement this algorithm however (see section 2.3.2).

The SEA Algorithm uses the triangle inequality to obtain a conservative estimate for the SAD at each search position. From the triangle inequality, the following inequality can be derived [38],

$$\Sigma_{x,y\epsilon A}|f_1(x,y)| - \Sigma_{x,y\epsilon A}|f_2(x+m_x,y+m_y)| \leq \Sigma_{x,y\epsilon A}|f_1(x,y) - f_2(x+m_x,y+m_y)| \tag{2.6}$$

If the conservative estimate, $\Sigma_{x,y\epsilon A}|f_1(x,y)| - \Sigma_{x,y\epsilon A}|f_2(x+m_x,y+m_y|$ for a search position, is higher that the current best SAD, then the SAD calculation for that position need not be performed. The SEA algorithm is extended in [49]. A closer estimate to the SAD is achieved by summing the conservative estimates of sub-blocks of the block being considered. This is beneficial because it increases the number of SAD calculations which can be skipped.

The SEA and EJO algorithms offer a method to reduce complexity when fixed

sized block matching is used. No investigation has yet been conducted into their effectiveness when variable block size matching is required. An efficient way of performing full search variable block size motion estimation [43] is to calculate the larger block SADs from the smaller block results. Thus, a more complicated determination of when a SAD result is and is not required is necessary, if support for variable block sizes is to be combined with either the EJO or SEA algorithms.

Vector Prediction and Early Termination

The extent to which the EJO and SEA algorithms reduce complexity is dependent on the order the search positions are evaluated. Ideally a search position close to the optimum should be evaluated first to ensure that a large number of SAD calculations are skipped. Vector prediction provides a means with which to find a position close to the optimum.

Vector prediction exploits the fact that both temporally and spatially adjacent blocks may have similar motion vectors. When variable block sizes are used the similarity between the optimum vectors for each block size can also be considered. By considering the range of possible vector candidates a good estimate for the current block vector can be found. In [44], every second block is assigned the same vector as the spatially adjacent block with the minimum SAD value. This is unusual, typically vector prediction is combined with search position reduction algorithms such as the diamond search. In [39], two algorithms are proposed which use vector prediction in combination with a diamond search pattern. One of the fast search algorithms implemented within the H.264 reference model, UMHexagonS [50], uses vector prediction in combination with both a hexagon based search and a localised full search.

The algorithms proposed in [39] and [50] also use early termination to reduce

complexity. When early termination is used, the motion estimation operation is terminated early if the current minimum SAD is less than a threshold value. Traditionally, the threshold value has been fixed. This is not ideal as the optimum threshold is dependent on a range of non-stationary factors, such as the degree of motion in a sequence and the noise present in a sequence. As a result adaptive thresholds have been used in [39] and [50]. In both cases the threshold value is generated from the minimum SAD values found for temporally and spatially adjacent blocks.

Algorithm and Architecture Co-Design

While compression performance and the number of computations are important other factors also influence the choice of motion estimation algorithm. How easily an algorithm can be implemented on the target hardware is also important. Despite its complexity the basic full search algorithm has remained a popular algorithm for hardware implementation due to its, minimal control overhead, high-level of data reuse and regular data flow. This has motivated the development of reduced complexity algorithms which have these desirable properties. The one dimensional search [32] is an example of such an algorithm. It reduces the number of search positions to be tested whilst maintaining the desirable properties of the full search algorithm.

Another example of a motion estimation algorithm specifically designed for hardware is the Global Elimination Algorithm (GEA) [51]. The GEA uses the conservative estimate, as used in the successive elimination algorithm, to reduce the number of candidates which need to be fully evaluated. This reduced candidate set is then evaluated using the SAD metric. Using the GEA the motion estimation hardware does not need to be able to calculate the SAD value for

every search candidate. It would need to be able to do this if the SEA algorithm was being implemented. Therefore, the GEA algorithm requires less hardware resources than the SEA algorithm. However, this resource saving is at the expense of compression performance.

In [52] an algorithm which is designed to minimise the power in ASIC implementations is proposed. It is based on the observation that in an ASIC implementation of a search position reduction algorithm, the majority of the power is consumed fetching the search data from the search memory cache (see section 2.3.2).

The power used by particular algorithms, when implemented in both ASICs and software, is studied in [53]. The set of algorithms chosen was full search, one dimensional full search, three step search, four step search, diamond search, modified log search, sub-sampling, and vector field sub-sampling. For ASICs [53] concluded that the full search algorithm consumed between 5 and 10 times more power that any of the other algorithms. It also concluded that when a small search range (8 pixels) is used there are negligible differences in the power consumed by the other algorithms studied. When a search range of 16 pixels is used [53] indicated that there are more significant differences in the power consumed by the various reduced complexity algorithms. Those algorithms which require a relatively large number of search positions to be tested (one dimensional full search, sub-sampling and vector field sub-sampling) consuming more than twice the power of the other reduced complexity algorithms. FPGA implementations were not considered in [53].

2.3.2 Full Pixel Motion Estimation: Architectures

In any video compression system, the architectures used for both full and fractional pixel motion estimation are critical. Commonly it is the algorithms and architectures used for motion estimation which have the greatest impact on the performance of the entire video compression system. In a pipelined encoder, for example, it is typically one of the motion estimation stages which determines Cp. This is because it is, in general, one of the motion estimation stages which requires the greatest number of clock cycles per macroblock [11][14].

Given this, is it is clear that the number of clock cycles required per macroblock, the macroblock latency, is an important parameter for full pixel motion estimation architectures. In [42] a distinction is made between macroblock latency and motion estimator throughput. It is argued in [42] that in a pipeline system, each stage cannot start operations for the next macroblock until all stages have finished operations for the current macroblock. Thus, the high throughput achieved by some full search motion estimation architectures, by performing operations on two or more macroblocks in parallel, will be difficult to take advantage of in a pipelined encoder. Other parts of the encoder would limit the overall encoder throughput. If a different encoder architecture was used it would be possible to to obtain a benefit from performing multiple motion estimation operations in parallel.

Achieving a low macroblock latency conflicts with the other motion estimator design goals. Most obviously with the resource minimisation goal. It also conflicts with the desire to minimise search memory bandwidth and search memory input bit width. For example, the full search motion estimator presented in [40] has a relatively high macroblock latency compared to other architectures that use a similar number of resources. However, it accepts search memory data in a serial

fashion and reads from each search memory location once only.

The principal benefits of minimising the search memory bandwidth are that, it minimises the power used reading data from the search memory and it allows the search memory to be more easily shared between the full and fractional motion estimators. The benefits of minimising the memory port bit width in an FPGA implementation are less obvious. The search memory is typically implemented using the embedded RAM blocks available on the FPGA (refer to section 2.3.2). Each individual embedded RAM has an output port size of 16, or more commonly 32 bits. Although smaller port sizes are supported, no implementation benefit is accrued using a smaller port size. Arguably there is a cost, the energy required for a 16 or 32 bit read operation will be incurred even when all 16 or 32-bits have not been read [54].

Resource utilisation is also an important metric for a motion estimator architecture. In general, the higher the resource utilisation the higher the throughput for a given set of resources. A full search implementation achieving a near 100% utilisation requires more complicated sequencing. Typically, the snake scanning order, instead of the raster scan order, is adopted to achieve near 100% utilisation [43].

Full Search Architectures

As previously mentioned, a large number of full search architectures have been proposed. In this thesis, the focus is on implementations which perform the variable block size motion estimation operation required by the H.264 standard. A more general review is given in [42].

In [43], full search architectures are classified into inter architectures and intra architectures. Inter architectures are where each processing element calculates

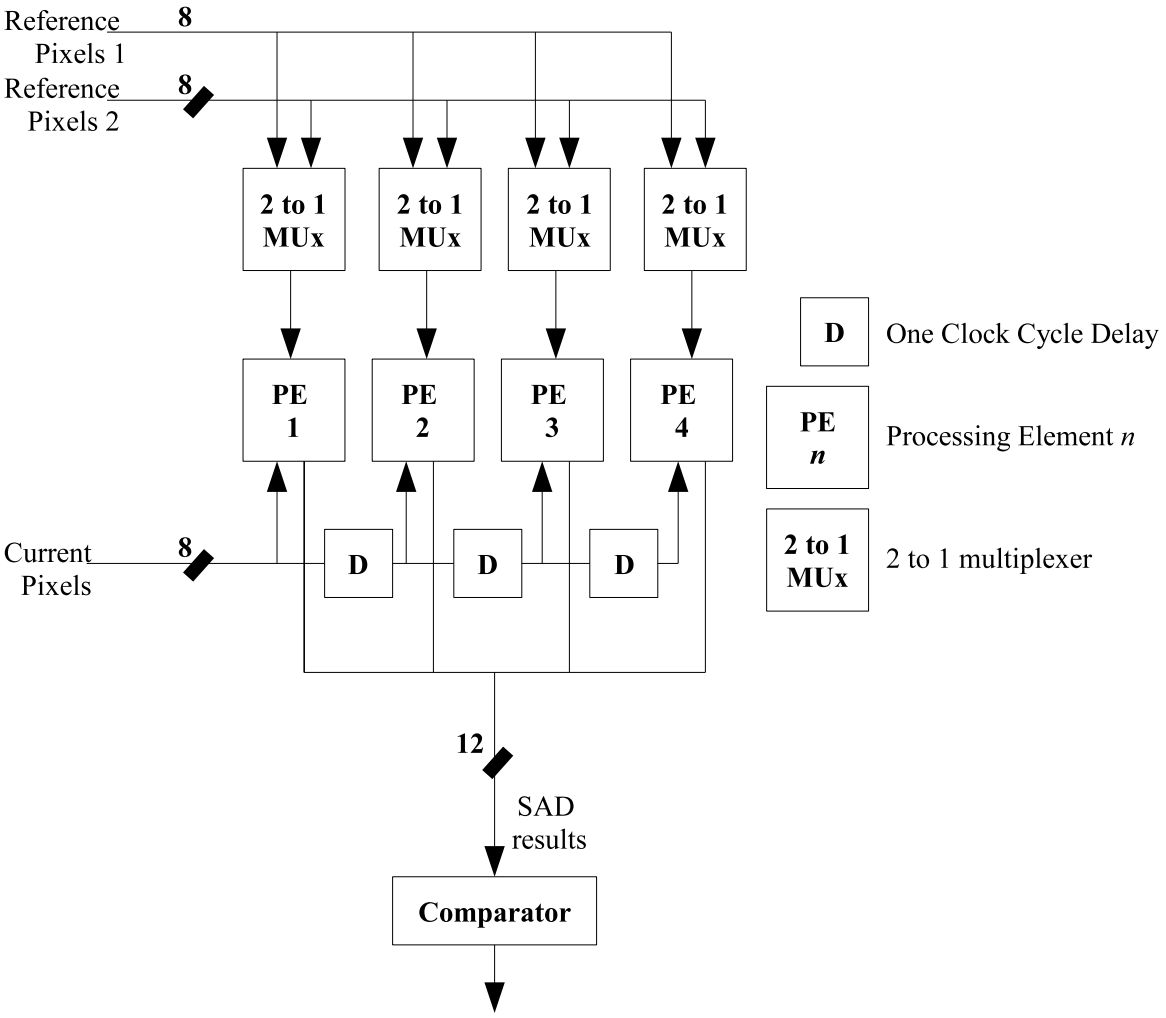

Figure 2.7: Example inter full search architecture

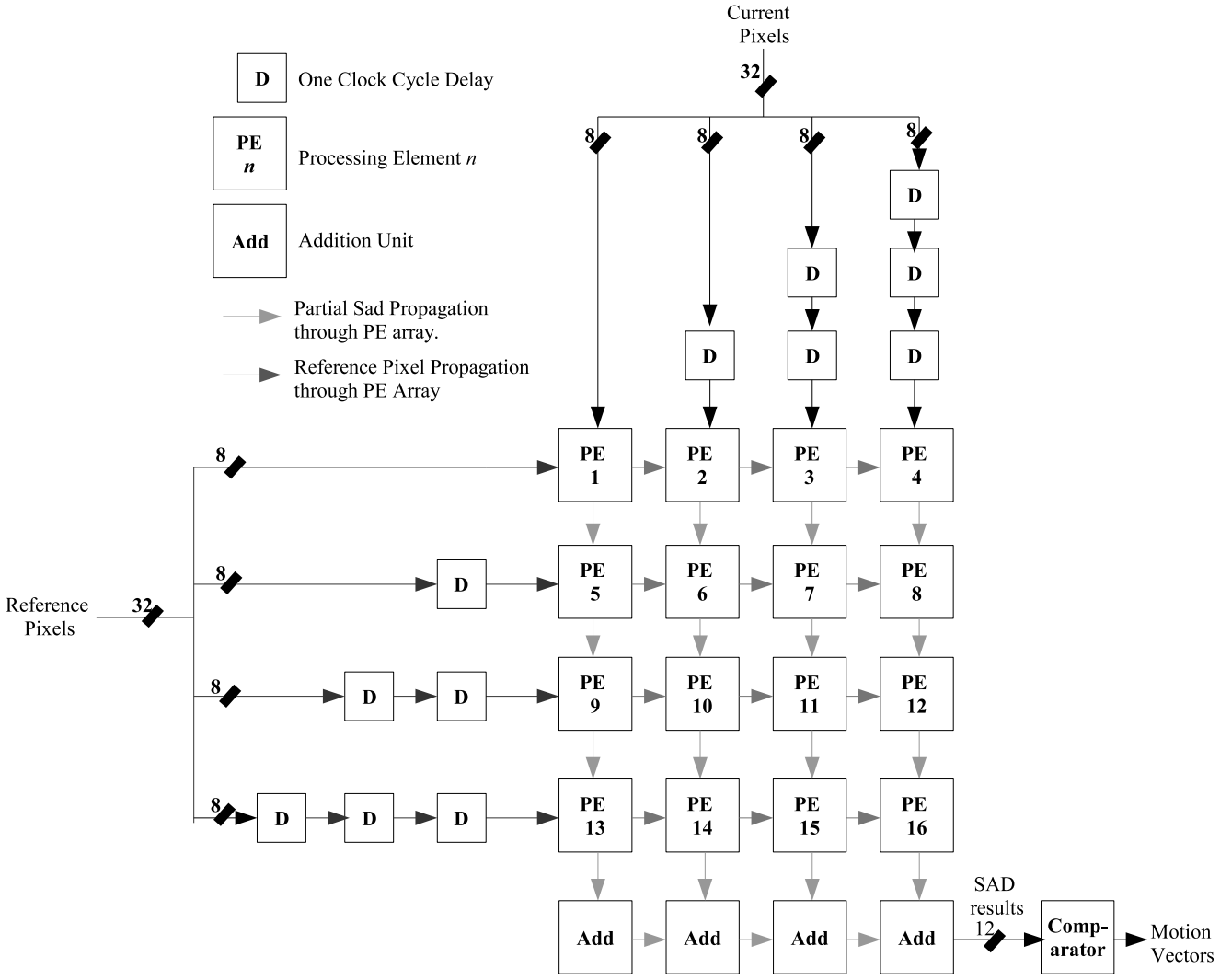

Figure 2.8: Example intra full search architecture

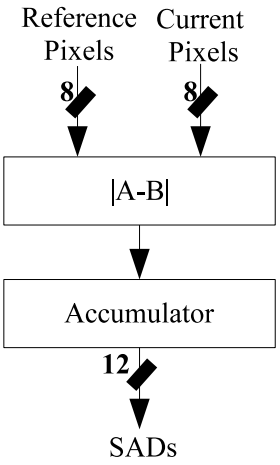

Figure 2.9: Inter search architecture processing element

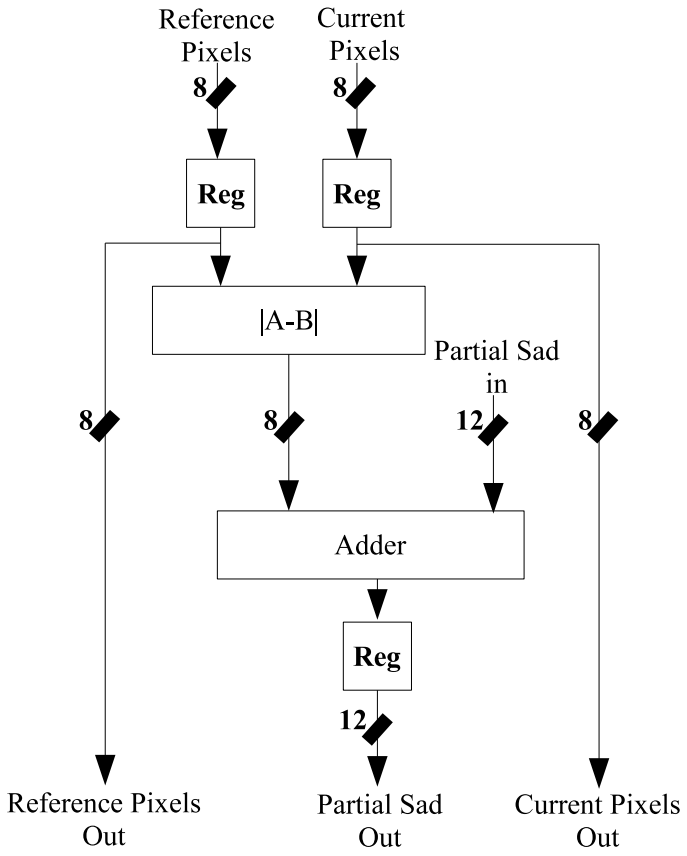

Figure 2.10: Intra search architecture processing element

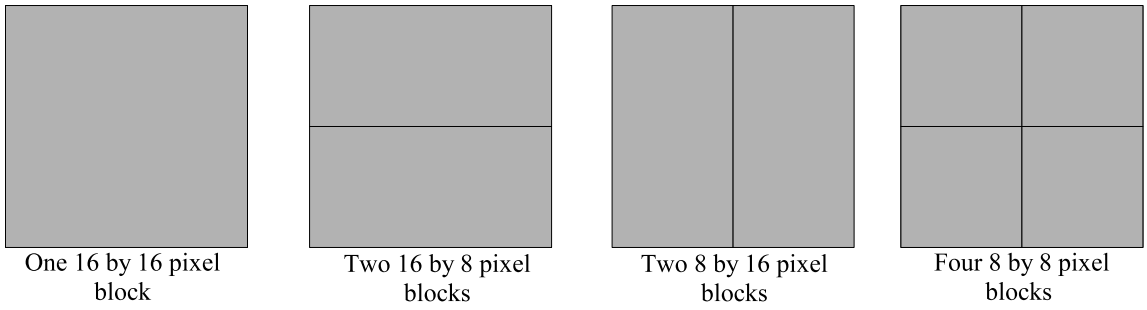

Figure 2.11: Different ways a macroblock can be sub divided for the H.264 motion estimation operation

the SAD for specific search positions. Intra architectures are where each processing element calculates all the absolute differences for a specific current pixel. An example of an inter architecture is shown in Figure 2.7. An example of an intra architecture is shown in Figure 2.8. The processing elements used by each architecture are shown in Figures 2.10 and 2.9

Note that to simplify Figures 2.7 and 2.8 the architectures use a smaller number of PEs than would be used in practice. For the inter architecture shown in Figure 2.7, sixteen processing elements are typical used [55] [56]. The intra architecture shown in Figure 2.8 would require two hundred and fifty six processing elements to support the typical macroblock size of 16 by 16. The inter architecture shown is a one dimensional architecture. It calculates the absolute difference values required for a search position sequentially, on a row by row basis. The intra architecture shown is a two dimensional architecture. All the absolute difference values required for a search position are calculated in parallel. Due to the parallel processing achieved by two dimensional architectures they have a much lower macroblock latency than one dimensional architectures. This is at the cost of a much greater resource usage however.

In H.264 a number of different motion estimation block-sizes and block-size

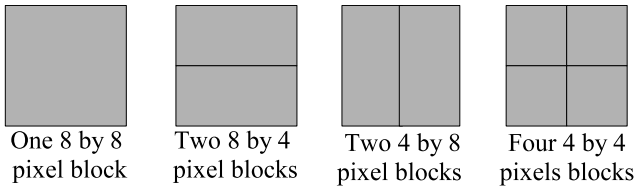

Figure 2.12: Different ways a 8x8 block can be sub divided for the H.264 motion estimation operation

combinations are supported. One motion vector can be used for the entire macroblock, or a macroblock can be sub-divided with separate motion vectors used for either two 16 by 8 pixel sub-blocks, two 8 by 16 pixel sub-blocks or four 8 by 8 pixel sub-blocks as shown in Figure 2.11. If the 8 by 8 mode is used, then each 8 by 8 pixel block can be further sub-divided into two 8 by 4 pixel sub-blocks, two 4 by 8 pixel sub-blocks or four 4 by 4 pixel sub-blocks as shown in Figure 2.12. Thus, for H.264 full search variable block size motion estimation a total of 41 motion vectors must be determined for each macroblock.

The method used in all full search VBSM implementations to date has been to calculate the larger sub-block SAD values from the SAD values of the appropriate smaller sub-blocks [43]. For inter architectures, such as that shown in Figure 2.7, it is predominantly modifications to the processing elements which are required for variable block size support. For example, the variable block size motion estimator presented in [56] uses the same basic architecture and data flow as the architecture shown in Figure 2.7. However to support variable block sizes the processing elements used are substantially more complex. The processing elements in [56] use 2 adders, fourteen registers with lengths between 12 and 16 bits, and 8 multiplexers. This compares to the one adder and one 16-bit register used when variable block sizes are not supported. In addition the architecture presented in [56] uses 13 comparators to determine the best motion vectors for

all the various sub-blocks.

In intra architectures, the summation operations required can take place internally within each processing element (a systolic array architecture) or externally from the processing element array. To support variable block sizes some external summation operations are required because the SAD values for each separate 4x4 sub-block needs to be available. The architecture shown in Figure 2.8 for example, requires the 16x16 processing element array used for fixed block size motion estimation to be split into 16 separate 4x4 arrays. In [43] an intra architecture, SAD tree, is proposed where no summation operations occur within the processing element array. Instead an external adder tree is used to sum all the absolute difference values produced. This allows all the absolute difference calculations for a search position to occur in the same clock cycle. As a result, the SAD values for all sub-blocks can be calculated in the same clock cycle. This avoids the need to store the SAD values of smaller sub-blocks prior to calculating the larger sub-block SAD values.

The FPGA implementation of intra architectures has been studied in [57]. Two intra architectures are compared, SAD tree and a systolic architecture similar to that shown in Figure 2.8. The systolic architecture requires more resources, using approximately 2000 more slices in the Virtex-2 FPGA used. Results given indicate that the systolic architecture is more power efficient than the tree architecture. This is primarily due to the SAD tree architecture having a lower resource utilisation than the systolic architecture. A snake scanning dataflow for the SAD tree architecture that achieves almost 100% resource utilisation was proposed in [43]. If this was used it is probable that the power efficiency of the SAD tree would be, at the least, comparable to the power efficiency of the systolic array architecture. Supporting the snake scan dataflow does require additional

resources however.

Both the SEA [58] and EJO [59] [60] algorithms have been implemented in hardware. The motivation for these implementations being to reduce the power consumed by the full search hardware. The SEA algorithm was implemented in [58] using two systolic arrays. One was used for the SAD calculations, the other for the conservative estimate calculations. To achieve a power saving, the appropriate part of the SAD array was disabled when it was determined that the SAD value for a particular search position was not needed. Results are given in in [58] indicating that a 50% power saving is achieved compared to when only a SAD array is used. However using two arrays consumes a significant number of resources.

In [59] and [60], the EJO algorithm has been implemented. Both use a one dimensional architecture. This allows the appropriate hardware to be disabled if after N rows have been summed the partial SAD is greater than current best SAD. In [59], results are given indicating there is a resource saving with their EJO architecture compared to a similar full search architecture. However, this is only through the use of a carry sum adder, an inefficient structure when implemented on modern FPGAs. Discounting this, there is clearly a resource cost to implementing EJO as the number of comparators required increases. This is evident in [60], where to support EJO an additional 720 comparators are used. Such a large number is required because [60] attempts to use the EJO algorithm with variable block sizes. Neither [59] or [60] indicate the power saved using the EJO algorithm. Results are provided in [59] indicating that signal transitions are reduced by over 50%.

All architectures require that the best motion vectors for all sub-blocks are determined in parallel. This is necessary. In an FPGA implementation the

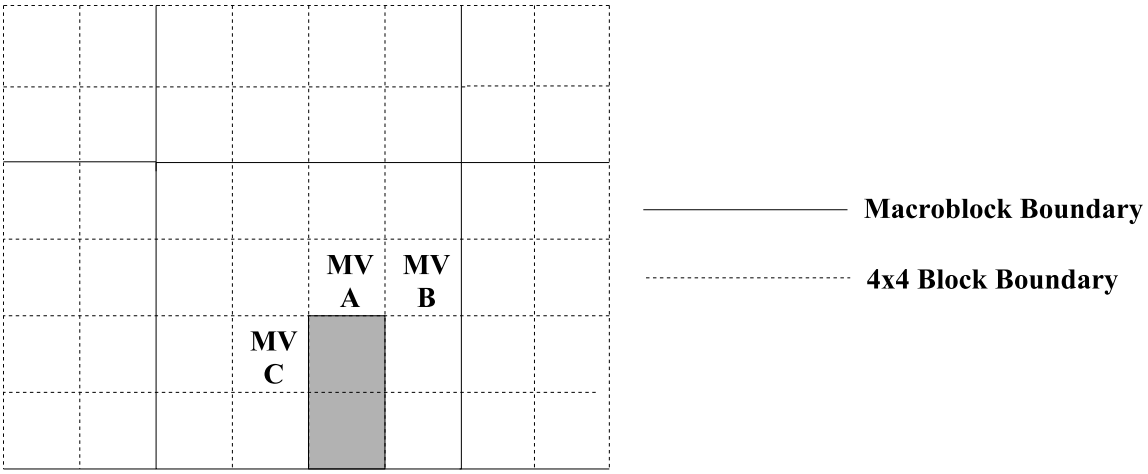

Figure 2.13: Block vectors used to determine $R(x, y, r)$ for shaded 4x8 block

motion estimator's performance would be severely limited if the motion vectors for all sub-blocks were determined in a serial fashion. The rate term $R(x, y, r)$ in equation 2.4 is dependent on the motion vectors chosen for surrounding blocks as indicated in Figure 2.13. Determining the best motion vector for each sub-block in parallel makes it impossible to determine the exact cost as defined in equation 2.4. In a pipelined encoder, the exact value of $R(x, y, r)$ cannot be determined for any sub-blocks because the motion vectors for the adjacent left macroblock will also be unknown. At the time they are required the fractional estimation process is still determining them. To enable the costs for candidate blocks to be determined [43] proposed to use motion vectors, A, B and D as shown in Figure 2.14 when calculating $R(x, y, r)$ for all sub-blocks.

Other Architectures

A large number of algorithm specific architectures have been proposed. For example in [61], an architecture for the three step search which achieves a high resource utilisation is proposed. Nine processing elements are used to calculate

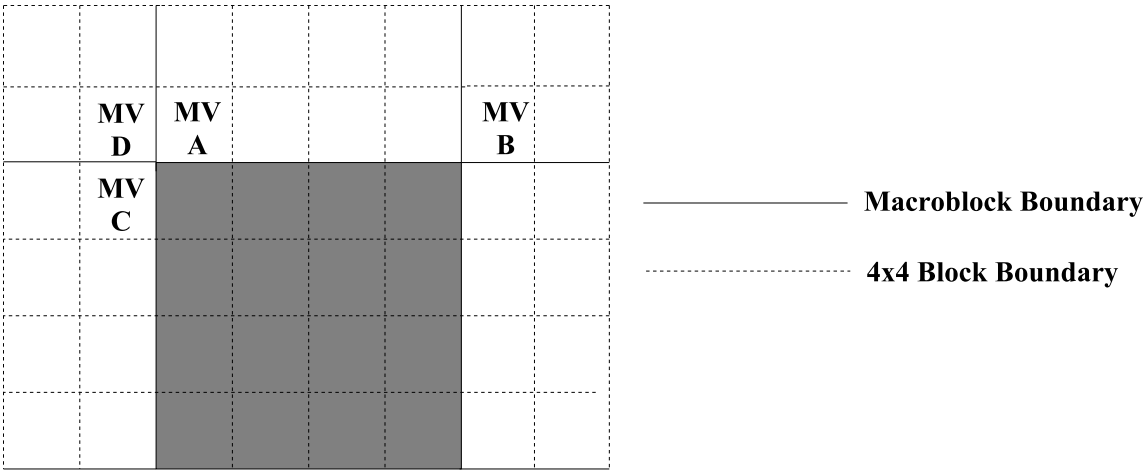

Figure 2.14: Modified predicted motion vectors A,B, and D used to determine $R(x, y, r)$ for all sub-blocks in shaded marcoblock

the SADs for the nine search positions in each step, with the reference data being supplied from 9 memories with 8-bit outputs. To ensure a high resource utilisation, without the need for a complicated interconnection network between the memories and processing elements, the search position each processing element is calculating alternates every 5 or 6 clock cycles. In addition the reference data stored in each memory is interleaved to ensure that reference data for each PE is available in every clock cycle.

Compared to other search position reduction algorithms, the three step search algorithm can be implemented relatively efficiently in hardware. As a consequence it has remained a popular candidate for hardware implementation. A H.264 full pixel motion estimation algorithm and architecture based on the three step search algorithm is proposed in [62]. This also uses an interleaved memory arrangement. However, because of the multiple steps used by the algorithm and the varying search patterns used in each step, a 100% utilisation is not achieved. This is typical of many fast search algorithm implementations. The irregular dataflow

associated with them makes achieving a 100% resource utilisation difficult.

Architectures which implement a range of search position reduction/vector prediction algorithms have been proposed. The motivation for these algorithms is flexibility. The choice of motion estimation algorithm does not need to be determined prior to implementation. The choice of algorithm being determined purely by the needs of the video compression application and by the types of motion expected in the sequences to be encoded. In [63], the architecture shown in Figure 2.7 is used as the basis for two configurable motion estimation architectures. In the first architecture, the current frame inputs shown in Figure 2.7 are replaced with a programmable interconnection network connected to 16 current frame memory banks. In the second architecture, the reference frame inputs shown in Figure 2.7 are replaced with a programmable interconnection network connected to 16 reference frame memory banks. In [64] and [65], flexible architectures targeted specifically at FPGAs have been proposed. In both architectures, the search positions to be evaluated are stored in a programmable memory and the SAD calculations are performed in a serial fashion. Thus, in both architectures the overlap between the reference frame pixels required for each search position cannot be exploited. This increases the search memory bandwidth required.

Search Area Data Reuse

The search area data required for motion estimation is generally stored in a cache as illustrated in Figure 2.15. This allows the search area data to be reused for subsequent motion estimation operations. This is beneficial because it reduces the bandwidth demands on the typically external, memory used to store reference frame data and it provides the motion estimator with faster access to the data it requires. Data reuse when the Full Search Motion Estimation Algorithm is used

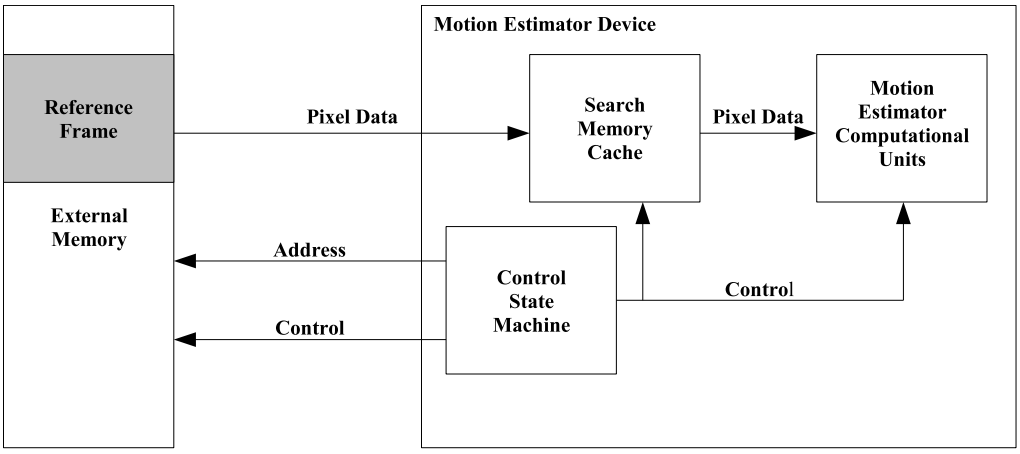

Figure 2.15: Typical memory hierarchy used in a motion estimation system

is studied in [66]. Four levels of data reuse are defined, A, B, C and D, with the degree of reuse increasing from levels A to D. In each case the architecture shown in Figure 2.15 applies. What differs in each case however, is the size of the search memory cache required and the amount of data that must be loaded from the external memory to the search memory cache to complete the motion estimation operations for a macroblock.

A and B are defined as candidate level reuse schemes, exploiting the overlap between the pixels required for adjacent candidate blocks. In the level A reuse scheme only the overlap, illustrated in Figure 2.16, between adjacent candidate blocks in a candidate row is exploited. To allow it to store the area to be reused between candidate blocks (marked in gray in Figure 2.16) the search memory cache in the level A reuse scheme must be able to store 240 bytes. The search memory bandwidth required is the highest of all the reuse levels defined in [66]. Assuming a search area of $+/-16$ pixels 25344 bytes must be loaded from external memory to complete the motion estimation operations for a macroblock. An example level A data loading schedule for a candidate row is shown in Figure 2.17.

In the level B reuse scheme, in addition to the overlap between adjacent can-

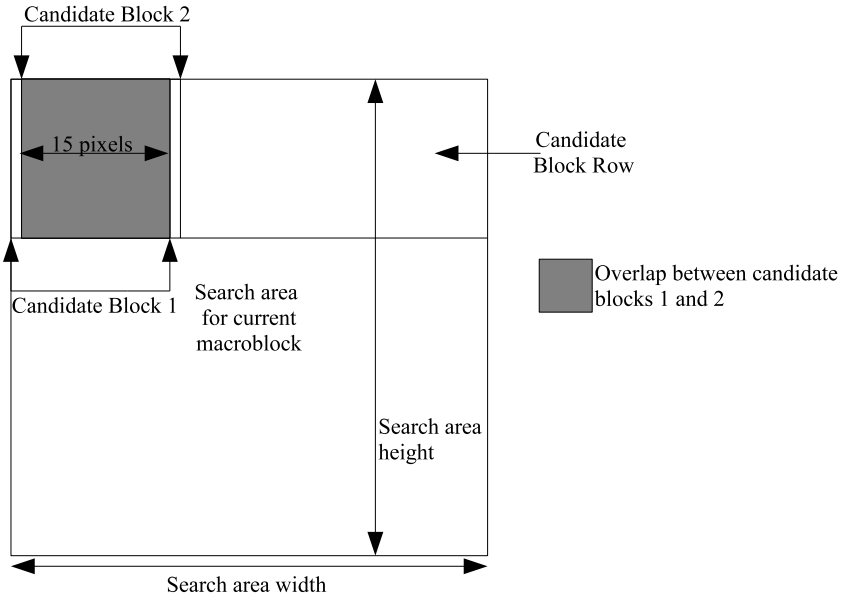

Figure 2.16: Overlap between the pixels required for adjacent blocks in a candidate row. Once candidate block one has been loaded only another 16 pixels must be loaded for candidate block two

Motion Estimation Computational Units	Candidate Block 1	Candidate Block 2	Candidate Block 3	Candidate Block 3	...	Candidate Block 32	Candidate Block 33
Data being loaded into search memory cache	256 pixels for candidate Block 1	16 pixels for candidate Block 2	16 pixels for candidate Block 3	16 pixels for candidate Block 4	...	16 pixels for candidate Block 32	16 pixels for candidate Block 33

Figure 2.17: Example data loading schedule for a candidate row when the level A reuse scheme is used

didate blocks, the overlap, illustrated in Figure 2.18, between adjacent candidate rows is also exploited. To do this the search memory cache must be able to store all the overlapped pixels between two adjacent candidate rows. For a search range of $+/-16$ pixels, this requires the search memory cache to have a size of 720 bytes. The benefit, compared to the level A reuse scheme, is that only 2304 bytes need to loaded from external memory to complete the motion estimation operations for a macroblock. An example level B data loading schedule for an

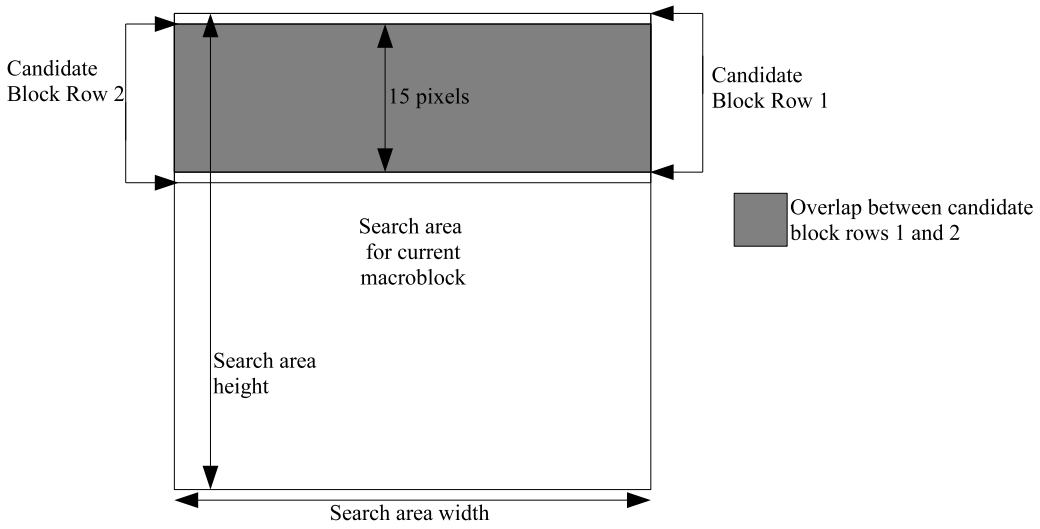

Figure 2.18: Overlap between the pixels required for candidate rows. Once candidate block row one has been loaded only another 48 pixels must be loaded for candidate block two, assuming a search area width of 48 pixels (search range +/- 16)

Motion Estimation Computational Units	Candidate row 1	Candidate row 2	Candidate row 3	Candidate row 4	...	Candidate row 32	Candidate row 33
Data being loaded into search memory cache	768 pixels for candidate row 1	48 pixels for candidate row 2	48 pixels for candidate row 3	48 pixels for candidate row 4	...	48 pixels for candidate row 32	48 pixels for candidate row 33

Figure 2.19: Example data loading schedule for a current macrablock when the level B reuse scheme is used

entire macroblock is shown in Figure 2.19

Reuse levels C and D are defined as block level reuse schemes. Instead of exploiting the overlap between candidate blocks, they exploit the overlap between the search area's of adjacent current macroblocks. In the level C reuse scheme the overlap, illustrated in Figure 2.20, between the search areas of adjacent current macroblocks in a single row is exploited. For a search range of +/-16 this requires a search memory cache of 1536 bytes. However only 768 bytes need to be loaded per current macroblock. An example data loading schedule using the level C

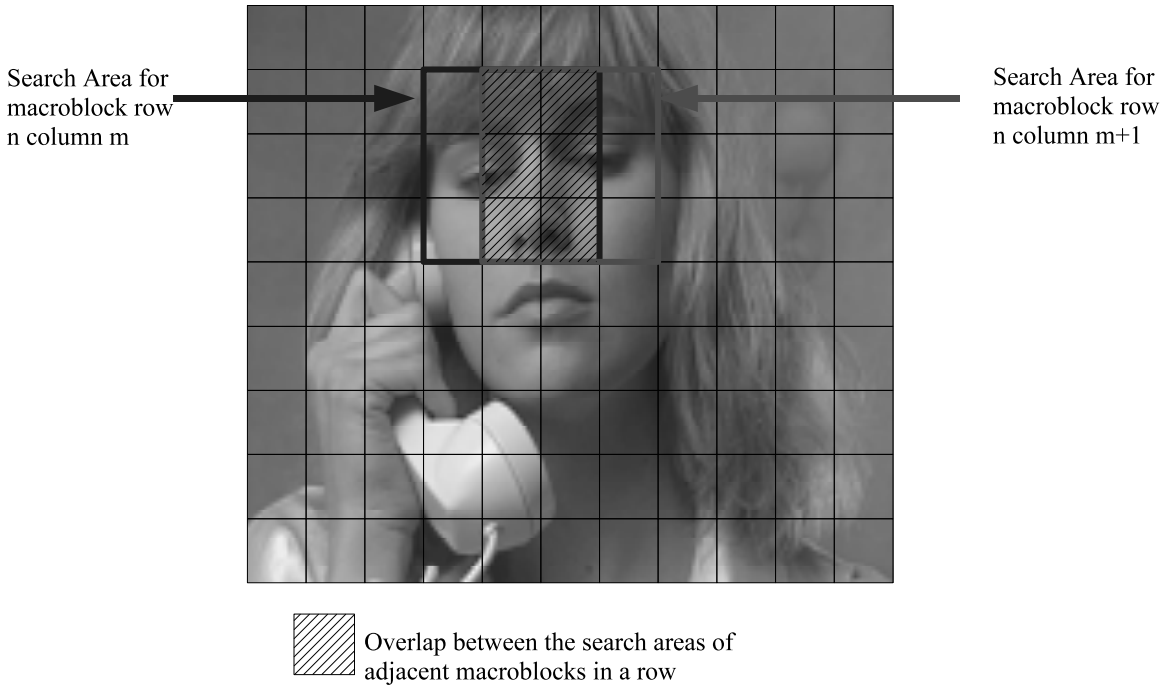

Figure 2.20: Overlap between the search areas of adjacent macroblocks in a single row

reuse scheme is shown in Figure 2.21.

In the Level D reuse scheme the overlap, illustrated in Figure 2.22, between the search areas of all current macroblocks in adjacent rows is exploited. This achieves the greatest level of reuse possible. Each reference pixel is only loaded once from main memory, with only 256 bytes having to be loaded per current macroblock. Do achieve this however, a large search memory cache is required, the size of which increases as the width of the frame to be encoded increases. For a search range of $+/-16$ and an image width of W pixels a search memory cache of $32 * W$ bytes is required. An example data loading schedule for the level D reuse scheme is shown in Figure 2.23.

It is assumed in [66] that the macroblocks in a frame will be processed in raster scan order. To reduce search memory cache size requirements whilst still

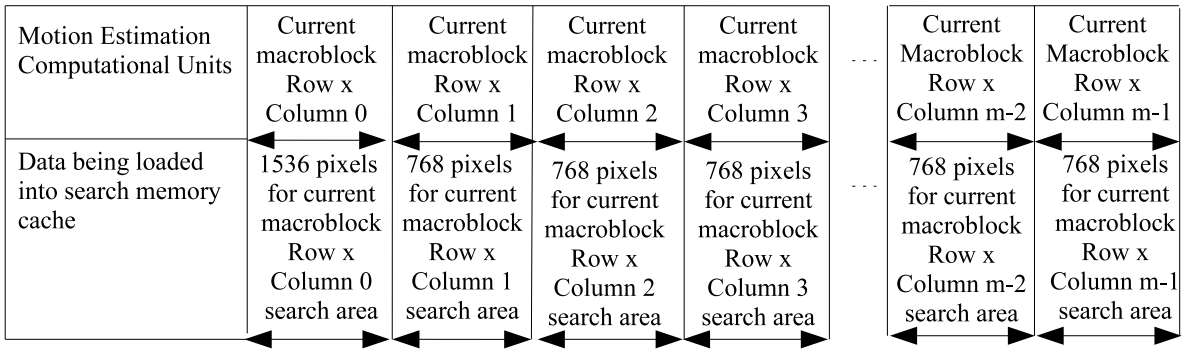

Figure 2.21: Example data loading schedule for a current macroblock row with m columns using the level C data-reuse scheme

allowing both the horizontal and vertical search area overlap to be exploited [67] proposed to use a zig-zag macroblock processing order. By using a zig-zag order, a better trade-off can be made between maximising data reuse and minimising the size of the search memory cache. However, using a zig-zag order complicates the design of the overall encoding system because the output bitstream, in general, must be in raster scan order.

Reuse levels A and B rely on the overlap between candidate blocks. As the candidate blocks to be considered differ depending on the motion estimation algorithm used, these reuse levels cannot be applied when the full search algorithm is not used. Reuse levels C and D rely on the overlap between search areas, consequently they can be applied even when the full search algorithm is not used. The search memory cache required is generally larger when the full search algorithm is not used. The memory sizes given above are based on the assumption that only the data to be reused needs to be stored in the search memory cache. For a full search implementation this assumption is valid. The predictability of the memory accesses associated with a full search implementation make it feasible that data can be loaded directly from external memory into the motion estimator

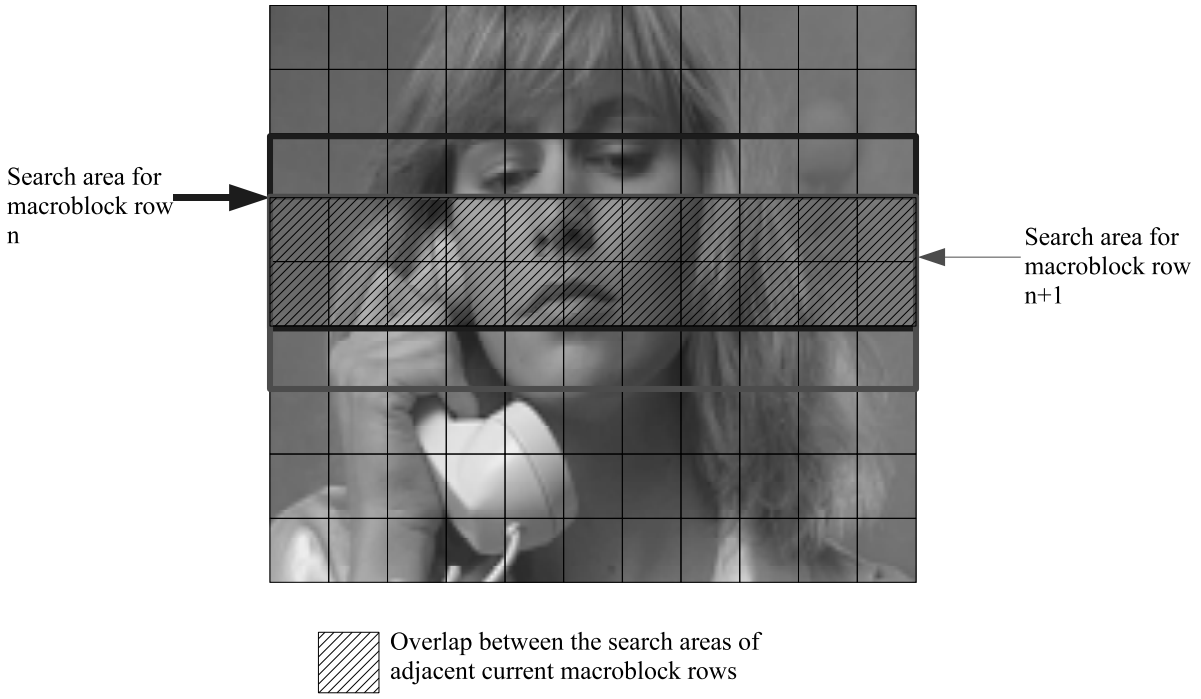

Figure 2.22: Overlap between the search areas of adjacent current macroblock rows

when required [66]. For other algorithms, the exact pixels required by the motion estimator will not be known in advance and in may cases the same pixels may be accessed a number of times during the execution of the motion estimation algorithm. As a result it is generally necessary to store the entire search area in the search memory cache if the full search motion estimation algorithm is not used.

With regard to FPGA implementations, the level C data reuse scheme has proved the most popular [10] [23] [14]. The MPEG-4 FPGA encoding system presented in [11] uses the level D data reuse scheme. The encoder in [11] further reduces memory bandwidth, by not writing a reconstructed macroblock to external memory when the macroblock is encoded using skip mode with a (0,0) motion vector. This is possible because under the conditions stated, each reconstructed

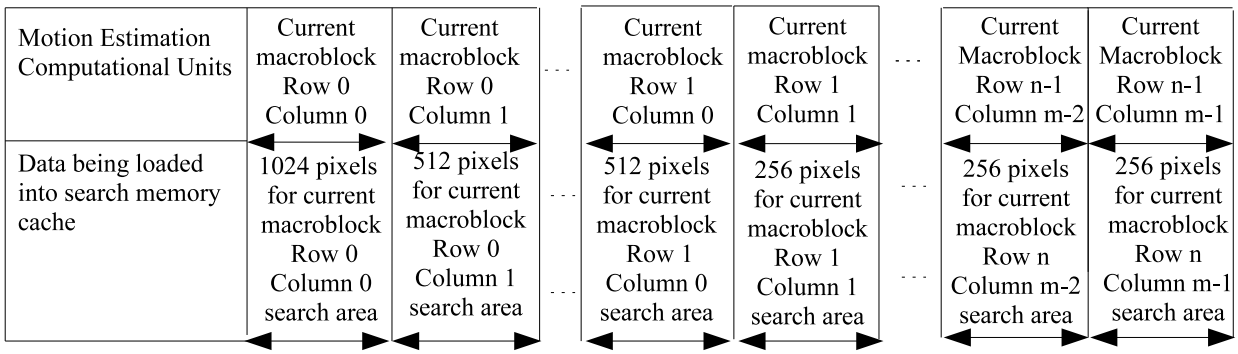

Figure 2.23: Example data loading schedule for an entire frame with n macroblock rows and m macroblock columns using the level-D reuse scheme

macroblock is identical to the co-located macroblock in the previous frame. For this optimisation to be practical each reference frame must be written to the same external memory location. Thus, it is only of use when the level D reuse scheme and one reference frame is used.

2.3.3 Fractional Pixel Motion Estimation

Intuitively, fractional pixel motion estimation improves compression performance by allowing a block's motion to be expressed more accurately. To do this, reference pixel samples at half pixel and, in H.264, quarter pixel locations are produced by an interpolation process. The exact interpolation process varies depending on which standard is being used. The complexity of the fractional motion estimation process is greater in H.264 than in previous standards, due to it employing a more sophisticated interpolation filter, and its support for both quarter pixel refinement and variable block sizes. A much larger number of positions require to be considered when quarter pixel refinement is supported as illustrated in Figure 2.24.

In H.264, the half pixel samples are generated using a six tap finite impulse

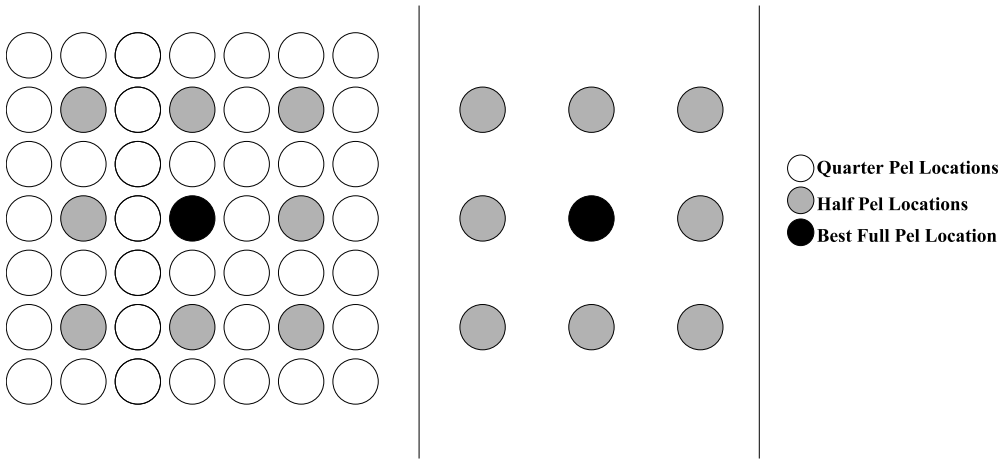

Figure 2.24: Fractional motion estimation search positions for a search range of one integer pel when supporting quarter(left) and half pel (right) refinement

response interpolation filter. This filter is designed to compensate for the aliasing which occurs in the digital video sampling process [68]. The quarter pixel samples are generated from half and full pixel samples using a simple bi-linear filter. To determine whether a half or quarter pel position offers a compression benefit equation 2.4 is used.

The standard algorithm used for the H.264 fractional estimation process is the full fractional search algorithm. This has been used in the JM reference model and by a number of fractional estimation architectures [19] [17] [18]. This is a two stage algorithm. In the first stage, the half pel positions surrounding the best integer search position are evaluated. In the second stage, the quarter pel positions surrounding the best half pel position are evaluated. Thus, for this algorithm 18 search points are required for evaluation. This requires significant computation. A separate interpolation and search operation must be performed on all 41 sub blocks. If a fast search algorithm is used for full pixel motion estimation the computation required for fraction motion estimation becomes more significant. This has motivated the development of fast search algorithms for the

fractional estimation process.

In [69], it was proposed to use a simpler bi-linear filter to generate the half pixel samples. However result showed that this reduced compression performance by nearly 0.5 dB. In [64], a 4 tap FIR filter is used. However, no results are given on the impact this has on compression performance. Note that while simpler filters can be used for the fractional estimation process, they cannot be used to generate the difference block. The 6 tap filter specified in the H.264 standard must be used at this stage to ensure that the reconstructed macroblock which is generated by the encoder matches that generated by the decoder. Search position reduction algorithms are another method which has been pursued to simplify the fractional estimation process. In [50], a diamond search pattern combined with vector prediction and early termination is proposed. Compared to the full pixel motion estimation process, there is less risk of a fast fractional estimation search algorithm becoming trapped in a local minimum. The monotonicity assumption holding more often because all the fractional samples are generated by an interpolation process [50]. No research has been conducted into applying the simpler distortion metrics discussed in section 2.3.1 to the fractional estimation process. Frequently, more complex distortion measures than the SAD are used. For example the Sum of Absolute Transformed Differences (SATD).

An unusual architecture for fractional estimation is proposed in [70]. Here the fractional and full pixel estimation stages are combined. This allows the SAD values of larger blocks to be summed from the SAD values of smaller blocks. This is not possible if fractional estimation is performed as a separate stage because in this case, each sub block will have a different search centre. The penalty for combining the two motion estimation stages is a limited search range and/or a large hardware requirement. The architecture presented in [70] only has a search

range of +/- 4 integer pixels, requires more than 28000 four input LUTs and uses 23 embedded RAMs to supply the reference pixels at the required rate.

More conventional architectures are presented in [19] [17] [18]. All implement the Full Fractional Search Algorithm and are designed primarily to support 4x4 block estimation. Larger block sizes are supported through decomposition into their constituent 4x4 blocks. To improve efficiency, work in [18] decomposes blocks with a width of 8 or more into 8x4 blocks. This allows greater reuse of the interpolated samples generated. However, it requires the estimator to support two pipelines, one for 4x4 and 4x8 block sizes, and one for all other block sizes. The other notable distinction between the architectures is how they implement the two algorithm stages. In [17] and [18], the same hardware is used to perform both the half and quarter pixel estimation. While this reduces the area and allows a high resource utilisation, it will impact on power because it requires the half pixel interpolation operation to be performed twice. In [19], registers are used to store the results of the half pixel interpolation process, with separate hardware for the half and quarter pixel estimates.

2.4 Intra Prediction

In total there are 22 intra prediction modes defined in the H.264 standard, 9 4x4 intra prediction modes, 9 8x8 prediction modes, and 4 16x16 prediction modes [1]. Each mode uses pixels above and to the left of the 16x16, 8x8 or 4x4 block being considered. The adjacent pixels being used to form a predicted block which is compared against the block being considered (Figures 2.25 and 2.26). The determination of the best intra prediction mode may be considered as part of the rate distortion optimised mode decision process discussed in section 2.5. If it is

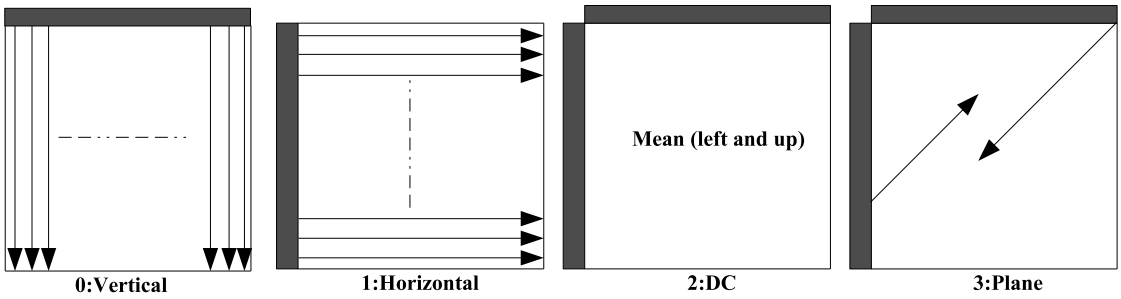

Figure 2.25: 16x16 intra prediction modes

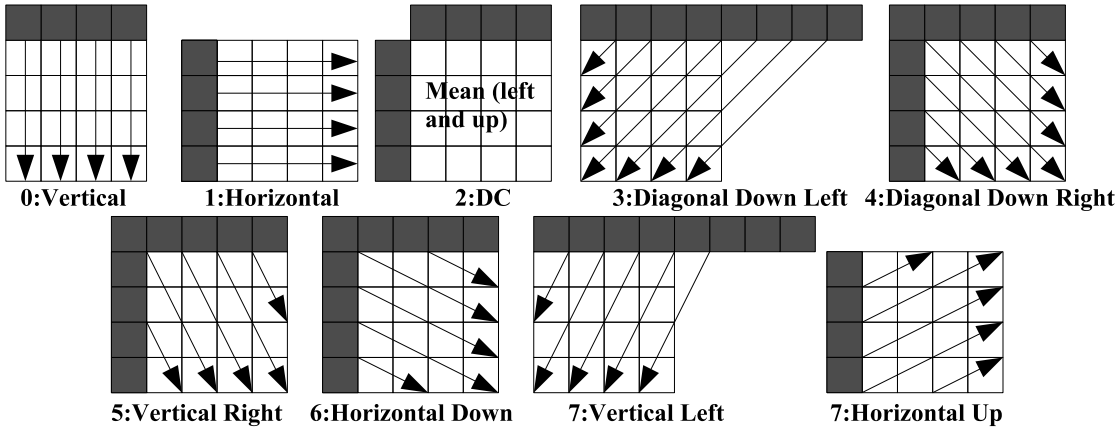

Figure 2.26: 4x4 intra prediction modes. The 8x8 intra prediction modes are similarly defined

not, a cost measure similar to that in equation 2.4 can be used to determine the best intra prediction mode.

Ideally, the best intra prediction mode should be determined in the same pipeline stage as the fractional pixel motion estimation operation. This allows the intra/inter mode decision to be made prior to the transform pipeline stage. However, as shown in Figure 2.27, a number of reconstructed pixels required for 16x16, 8x8 and 4x4 intra prediction will be unavailable in this pipeline stage.

If the best intra prediction mode is determined in the same pipeline stage as the transform and inverse transform, all the pixels required for 16x16 intra

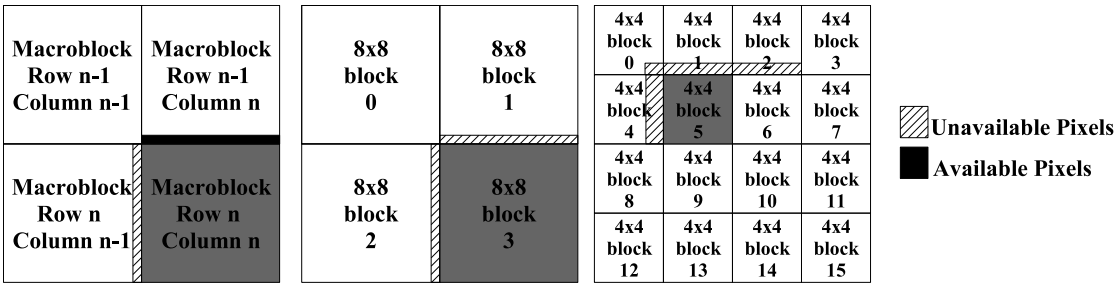

Figure 2.27: Unavailable reconstructed pixels required for (from left) 16x16, 8x8 and 4x4 intra prediction

prediction will be available. For the 8x8 and 4x4 intra prediction modes, only the top left sub-block will have all the reconstructed pixels it requires available initially. The pixels required for the other sub-blocks must be produced through the transform/inverse transform process. This forces the intra prediction, forward and inverse transform processes to be tightly coupled, restricting their ability to operate concurrently. It also implies that multiple transform operations will be required, one to determine the best 4x4 intra prediction mode, one to determine the best 8x8 prediction mode (assuming 8x8 intra prediction is supported), and one for the mode actually chosen for the macroblock being encoded.

The simplest solution to this problem is to use input pixels instead of reconstructed pixels when determining which intra prediction mode to use [69] [14]. This solves the data availability issue. Compression performance can be degraded, however, particularly for sequences with minimal temporal redundancy. An alternative approach is to optimise the sub block processing order to maximise the period during which the intra prediction and transform hardware can operate concurrently [71]. This allows some resource savings but still requires multiple transform operations to be performed. If reordering is used with a fast intra/inter mode decision algorithm then the need for multiple transform operations can also

be removed.

2.5 Mode Decision

In H.264, there are a greater number of modes which can be used to compress a macroblock than in previous standards. If the determination of which spatial prediction mode to use is considered as part of mode decision process there are 592 different mode combinations to choose from for I-slices (assuming 8x8 intra prediction is not supported). Additional modes must be considered when P-Slices are used (INTRA 16x16, INTRA 4x4, INTER 16x16, INTER 16x8, INTER 8x16 and INTER 8x8). For the INTER 8x8 mode 4 modes must be considered for each 8x8 sub-block (INTER 8x8, INTER 8x4, INTER 4x4 and INTER 4x4). This compares to the 3 modes available in MPEG-2, and the 4 available in H.263/MPEG-4 [8].

The rate distortion optimised (RDO) mode decision algorithm implemented within the H.264 reference software is computationally expensive. The distortion between the encoded and unencoded macroblocks and the rate for that distortion must be measured for each mode being considered. This requires each macroblock to be encoded multiple times. As a result of the RDO mode decision algorithm's complexity, it has not been frequently used in H.264 hardware implementations. Instead less complex mode decision algorithms have been used. The simplest of which is to use the cost measures calculated in the intra prediction and motion estimation stages to determine which encoding mode to use. Compared to the RDO algorithm this offers less compression performance but is substantially less complex.

Efforts have been made to develop mode decision algorithms which reduce

complexity further. In general the method used to do this is to make the mode decision prior to performing all the calculations required by the motion estimation and intra prediction processes. In [72], it is proposed to determine the mode used for some marcoblocks partly on the modes chosen for spatially adjacent macroblocks. This is based on the assumption that the modes chosen for adjacent macroblock's are correlated. While reducing complexity [72], still requires either the 4x4 intra prediction process, the 16x16 intra prediction process or the motion estimation process for a specific block size to be performed for every macroblock.

Another proposal in [73] determines which mode class, inter or intra, to exhaustively search. To determine whether the intra or intra mode class should be used, work in [73] uses features representative of both the spatial and temporal redundancy. The spatial feature used is the minimum SATD of a subset of the 4x4 intra prediction modes. The temporal features used are the SATD and motion vector length of best motion vector candidate as determined by the vector prediction algorithm proposed in [39]. Using these features, a Bayesian cost based decision is made on which mode class to use.

In [74], an algorithm is developed to predict when the SKIP encoding mode offers the best compression performance. When the algorithm predicts that the SKIP mode offers the best compression performance, neither the motion estimation and intra prediction processes need to be performed. The algorithm developed in [74] uses a model to predict the distortion and rate for a macroblock when it is encoded in the normal fashion (i.e by performing both motion estimation and intra prediction). The models are based on the following assumptions,

- The best mode for a macroblock and its associated distortion will be the same as that of the co-located macroblock in the previous frame

- The required rate for the best mode will be half that of the co-located macroblock in the previous frame.

Using these assumptions the cost for best mode can be determined and compared against the cost for the skip mode. Result are given in [74] showing that using the proposed algorithm reduces encoding time in a software encoder by between 30% and 70%.

Only work in [23] has used a mode decision algorithm which allows the skipping of some of the required motion estimation and intra prediction calculations in a pipelined encoder implementation. In general, such algorithms offer the potential to reduce the power used in a pipelined encoder. The resource savings offered by such algorithms are limited because the encoder still needs to be able to perform the full motion estimation and intra prediction operations when required. Although, as discussed in section 2.4, using a fast mode decision algorithm which predicts when either the 8x8 or 4x4 intra prediction operations need to be performed can reduce the number of transform operations which need to be performed. Consequently, such algorithms have the potential to reduce the resources used by the transform component.

2.6 Transform

The transform stage within DCT/DPCM based compression de-correlates the video image data, facilitating compression. In most standards apart from H.264 the Discrete Cosine Transform (DCT) is used to do this because it has been shown to be a good approximation of the optimal Karhunen Loeve Transform (KLT) for natural video images. The transform is also used to de-correlate the residual images formed as a result of motion estimation/compensation. In [75] it

was shown that the KLT for a motion compensated difference image is identical to the KLT for the original video image and that the DCT remains a good approximation of the KLT for motion compensated difference images.

Within H.264, a 4x4 integer transform, derived from the DCT, is used. This is in contrast to previous standards which have used an 8x8 DCT. An integer transform is used to remove the possibility of the encoder and decoder reference images differing due to rounding errors in the encoder and decoder DCT implementations. A 4x4 transform is justified because the spatial and temporal prediction which occurs before the transform stage in H.264, negates to a large extent the correlations between each 4x4 transform block [76]. A new revision of the H.264 standard does provide support for an 8x8 integer transform [1]. The integer transform used in H.264 has comparable performance to the normal DCT transform [20]. In addition, it is simpler to implement because it does not require any multiplication operations. As discussed in section 2.4, the performance requirements placed on the transform and inverse transform implementation are dependent on the mode decision algorithm used and on the intra prediction architecture used. Within a pipelined encoder, the transform component's performance may not be critical. In the encoder described in [23] for instance, the vertical and horizontal transforms required are performed sequentially with one operation occurring per clock cycle. This gives a throughput of less than one pixel per clock cycle. The impact on overall encoder performance is masked however by the performance of other parts of the encoder implementation. This encoder uses a fast mode decision algorithm. As a result, the encoder only needs to perform one complete transform operation per macroblock.

If the encoder is specifically targeted at larger frame rates and resolutions a higher performance transform architecture may be needed. In [14], a distributed

architecture is used to implement the transform operations. Each processing element calculates the transformed value for one pixel in a 4x4 block. This is an unusual design which requires the output values to be multiplexed onto a data bus prior to quantisation. More common is to split the transform operation into its vertical and horizontal components, as in [23], but perform multiple transform operations per clock cycle [77] [78]. Throughput is further increased in [78] by pipelining the vertical and horizontal transform operations. Split architectures require a transpose memory to sequence the data appropriately prior to the second transform operation. In [79], the vertical and horizontal operations are combined to realize an architecture which does not require a transposition memory.

2.7 Entropy Encoding

There are two forms of entropy encoding available for use in H.264, Context Adaptive Variable Length Coding (CAVLC) and Context Adaptive Binary Arithmetic Coding (CABAC). CABAC provides better performance, reportedly reducing the size of the encoded bitstream by between 9% and 14% compared to CAVLC [80]. However, it is relatively complex and for some applications the implementation costs outweigh the benefits. Therefore, the lower complexity CAVLC entropy encoding mode is also supported by the H.264 standard.

2.8 Loop Filter

As previously stated the loop filter is designed to remove any artifacts which may have been introduced as a result of the encoding process. A filtering operation is applied to the horizontal and vertical edges of each 4x4 luma and chroma block.

Various conditions effect the strength of filter which is applied to each edge. These include whether the macroblock is encoded using an intra prediction mode, whether the edge is on a boundary between macroblocks and the difference in the value of pixels on either side of the edge. The filtering operation is also dependent on the quantisation parameter used, with a filtering operation more likely to be applied when a higher quantisation parameter is used. These conditions are designed to ensure that the loop filter filters artifacts introduced by the encoding process but does filter true edges which exist in the image [81]. The strength of the filter can be adjusted by the encoder and can also be completely switched off by the encoder.

2.9 Summary

The hybrid DPCM/DCT framework has been used by the majority of compression standards. However, due to improvements to the various processes used within it, newer standards have still provided an improvement in compression performance. Key improvements incorporated into the H.264 standard include, variable block size motion estimation, multiple reference frames, an integer transform and intra prediction.

A pipelined architecture is the predominant architecture used to implement a video encoder in an FPGA. The key part of any encoder implementation is the algorithms and architectures used for motion estimation. . Secondly, due to the complexity associated with the motion estimation process, the motion estimation stages of a video encoder have the greatest influence on the overall throughput of a video encoder implementation.

2.10 Conclusions

Implementing a video encoder requires substantial memory and computational resources. Many of the key improvements incorporated into the H.264 standard, such as variable block size motion estimation, multiple reference frames, intra prediction and loop filtering have increased the memory and computational resources required. The resources required can be reduced, principally by reducing the complexity of algorithms used for motion estimation, intra prediction and mode decision. However, this reduces an encoder's compression performance. Balancing the trade-off between compression performance and complexity of the encoder implementation is key when designing a video encoder implementation. The optimum balance between complexity and encoder performance is very application dependent.

Using a flexible programmable solution, such as an FPGA, allows the trade -off between complexity and performance to be adjusted as required. More importantly, the algorithms used for motion estimation, mode decision and intra prediction continue to be updated, in order to provide better performance and/or lower complexity. Using a programmable solution ensures that any improvement in the algorithms available can easily be incorporated into an encoder implementation. In comparison incorporating new algorithms into an ASIC encoder implementation would be a difficult and expensive task.

As discussed in chapter 1 there are other programmable technologies which could be used to implement a video encoder. FPGAs, however, offer a number of advantages to a video encoder implementation. The encoder architecture can be determined by the needs of the application, instead of being determined by the structure of the programmable system. Secondly, FPGAs come in a range of

sizes, this ensures that FPGAs can meet the varying computational requirements of each video encoding system and that there is minimal wastage of computational resources. Finally, FPGAs are robust technology with a well understood development flow. Thus, using FPGAs is a lower risk that alternatives such as coarse grained reconfigurable arrays. FPGA solutions, whilst offering a number of advantages, can suffer from a high level of power consumption compared to other reconfigurable implementation technologies. This is principally a result of the large degree of reconfiguration supported by an FPGA. Computationally complex applications, such as H.264 video encoding, can exacerbate this problem, particularly if the FPGA implementation does not take advantage of some of the power saving features offered by the FPGA device. As discussed in chapter 1, for many video compression applications power consumption is an important factor.

Whilst FPGA device manufacturers have addressed power consumption at the device level. It is the contention of this thesis that, for the video encoding application, significant savings can be made at the application level. This is the focus for the majority of research described in the rest of this thesis.

.

Chapter 3

Field Programmable Gate Arrays

FPGA based designs have always consumed more power and area for a given clock rate than equivalent ASIC based designs due to the overhead associated with reconfigurability [82] [83]. Despite the reduced power consumption achievable, however, ASICs remain unattractive for all but the most high volume applications due to the prohibitive non-recurring engineering cost. In this chapter the basic architecture used by most modern FPGAs is introduced. The types of FPGA power consumption and the methods used to reduce each type are then discussed.

3.1 FPGA Architecture

A simplified diagram representative of a modern low cost FPGAs is shown in Figure 3.1. Such an FPGA contains four main elements, logic arrays, embedded SRAM, embedded multipliers, and Input/Output (IO) blocks. Each logic array is made up of clusters of logic elements as shown in Figure 3.2, with each logic element containing a SRAM or FLASH based Look-Up-Table (LUT) and a register.

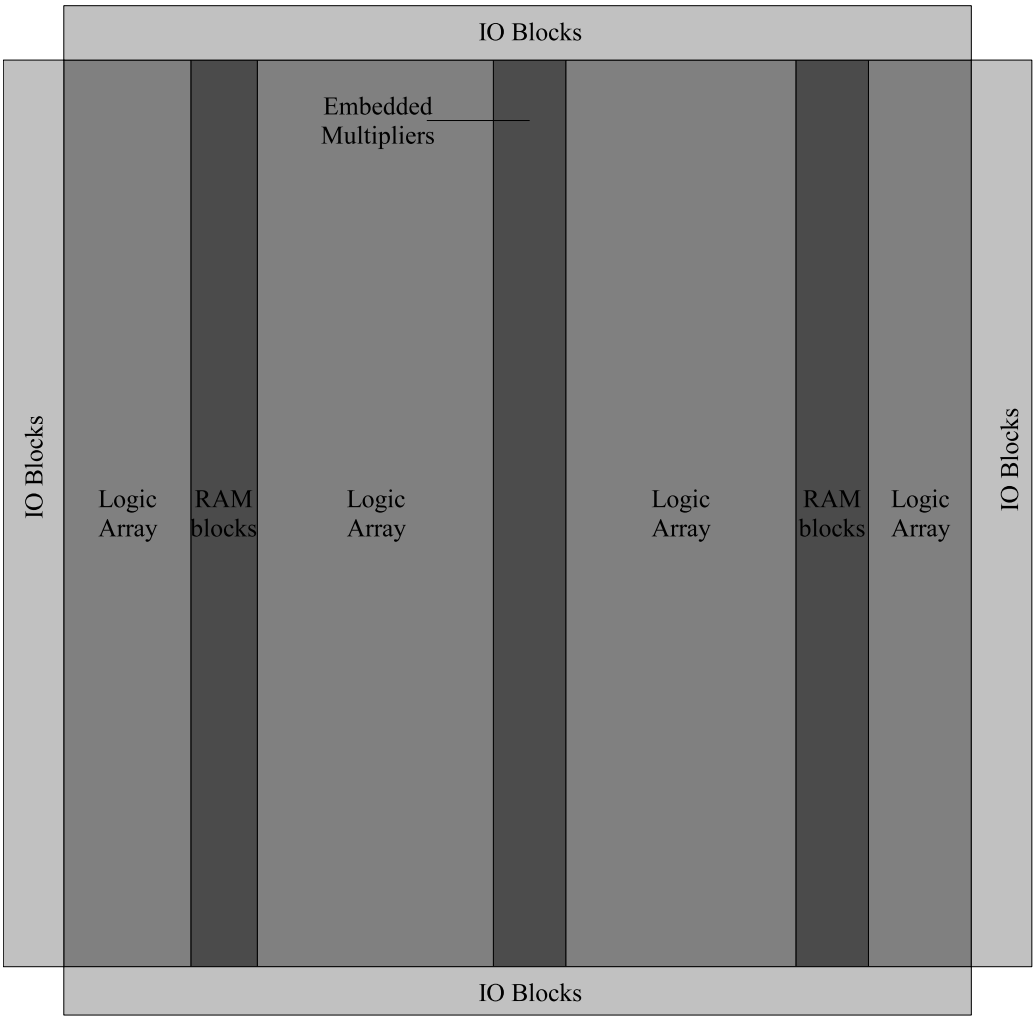

Figure 3.1: Generic block diagram of a modern FPGA

The various elements are connected together with a range of interconnect resources. As an example, consider the Altera Cyclone-2 FPGA. The clusters, or Logic Array Blocks (LAB), consist of 16 logic elements, with direct links between adjacent LABs in the horizontal direction, horizontal interconnect spanning 4 and 24 LAB columns and vertical interconnect spanning 4 and 16 LAB rows [84]. The embedded multipliers and embedded ram within a Cyclone-2 FPGA are connected to the same routing network as the LABs [84]. In addition to the general interconnect resources available, there are also dedicated interconnect

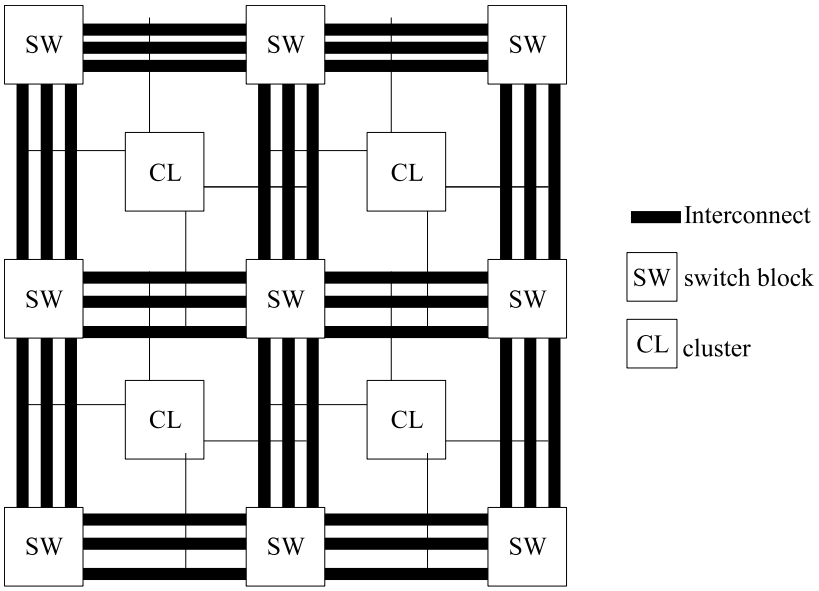

Figure 3.2: Diagram of an FPGA logic array

resources to support the propagation of the carry bit used in arithmetic operations and to support the distribution of clocks to the FPGA's synchronous elements.

The carry chain interconnect and logic allows the propagation of a carry bit from one logic element to the vertically adjacent logic element in the logic array. Including this addition logic and interconnect, enables the FPGA to perform basic arithmetic operations at a much faster rate than would be the case if it were not present. The clock distribution networks on an FPGA are specifically designed to provide a low skew path for the clock signals required by the synchronous elements present within an FPGA. In the low cost FPGAs used in this thesis these are all global clock distribution networks. Higher cost FPGAs, such as the Xilinx Virtex-5, have both global clock networks, routed through the entire FPGA, and localised clock networks, each of which is routed through a rectangular section of the FPGA device.

Figure 3.3 shows the clock distribution network present on a Cyclone-2 FPGA. It consists of 16 global clock networks which feed clocks to each LAB row on the

FPGA. Each LAB row clock net then feeds upto 6 of the available global clocks to the synchronous elements on that LAB row and to a section of the IO blocks present on the device. Buffers are used to ensure that each LAB clock net is only active when necessary. If a particular LAB row contains only unused synchronous elements no clock signals are fed onto the LAB row clock nets. This is common, most modern FPGAs implement similar functionality in order to reduce the power used distributing clock signals through the FPGA device [84] .

The clock control blocks contain multiplexers which allow the clocks fed onto the global clock networks to be dynamically selected after the FPGA is configured. The clocks supplying the clock control blocks can come, directly from an external source, from a phased locked loop (PLL) output, or be generated internally within the FPGA device. The Cyclone-2 FPGA includes phased locked loop(PLL) circuits to provide the ability to compensate for the delays introduced by both the on and off chip clock distribution networks. Using the PLL circuits ensures that all the clocks present in a system are correctly aligned. The PLL circuits also provide the ability to adjust the frequency of the incoming clock, depending on the requirements of the FPGA design. Circuits providing the same function are present in the majority of other modern FPGAs. For example, the Xilinx Spartan-3 FPGA includes delay lock loop (DLL) circuits, which while differing in their implementation, provide similar functionality to the PLL circuits implemented in the Cyclone-2 architecture [85].

Each IO block in an FPGA is connected to a pin, and can be configured to either provide an FPGA input or an FPGA output. In the Cyclone-2 architecture, they consist of a tristate buffer and 3 registers, an input reqister, an output register and a tristate enable register as shown in Figure 3.4. Use of the IO registers is optional, configuration multiplexers not shown in Figure 3.4 allow the

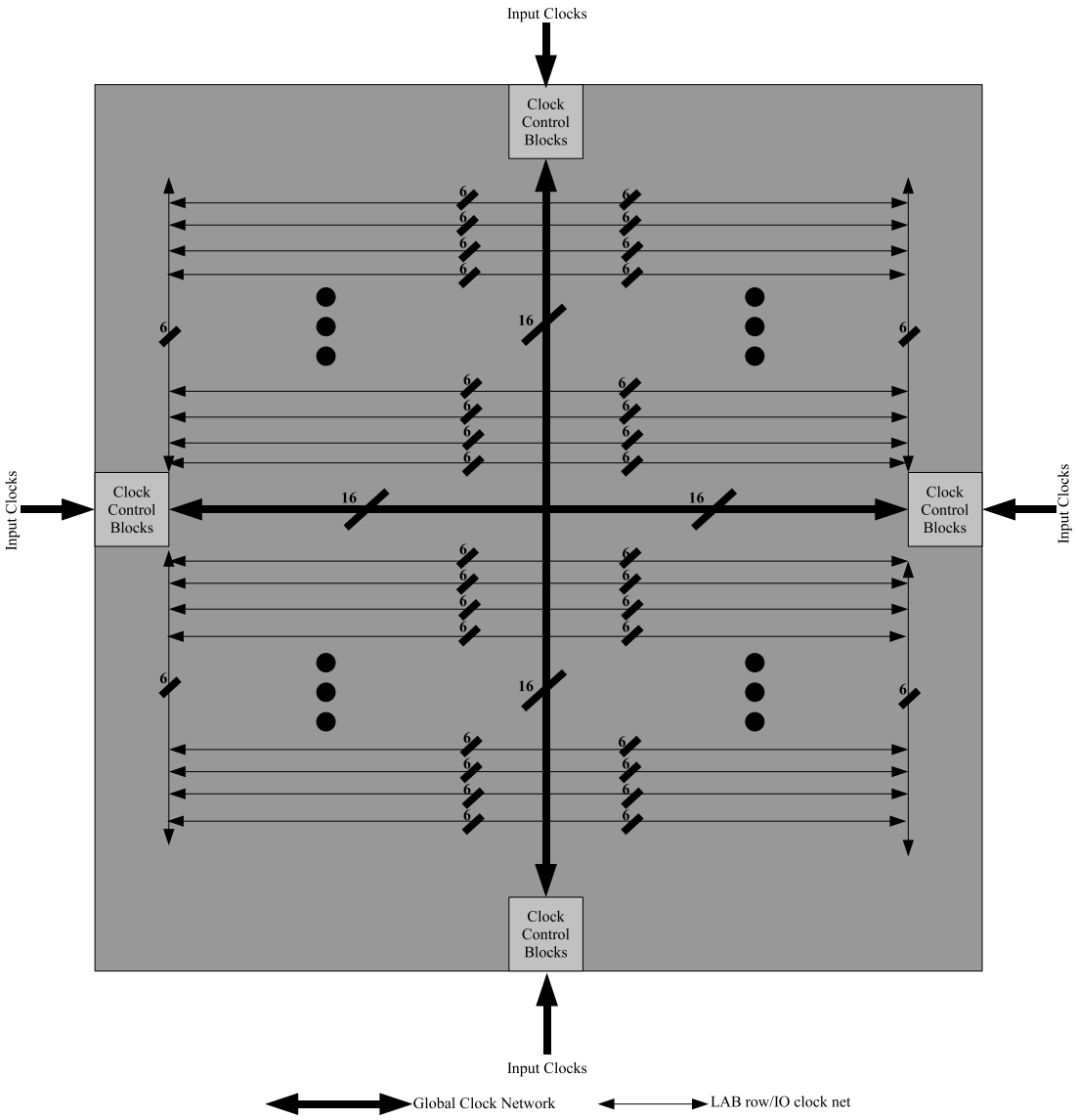

Figure 3.3: Clock Distribution in Cyclone-2 FPGAs

IO registers to be bypassed if necessary. Using the IO registers however, ensures that the maximum timing margin achievable is available when communicating with other devices within a system.

The embedded memory and embedded multipliers present in modern FPGAs reduce the performance gap between an FPGA and an ASIC, particularly in terms

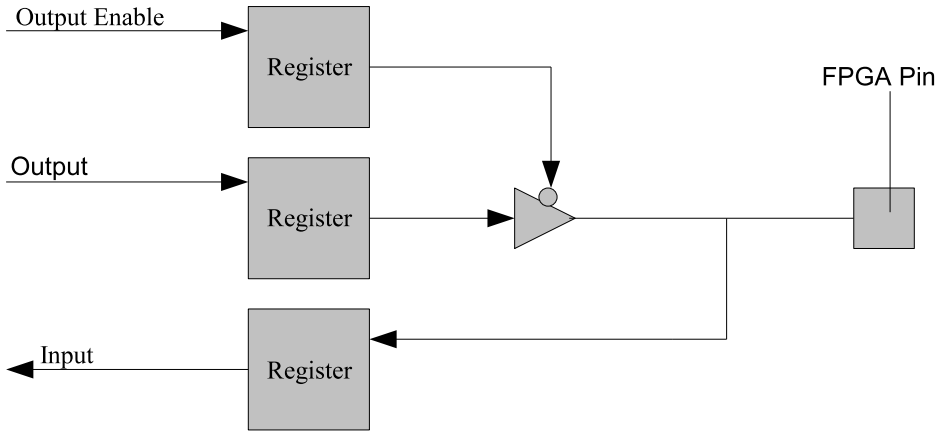

Figure 3.4: Simplified diagram of a cyclone-2 IO block

	Spartan 3 [85]	Cyclone 2 [86]	Cyclone 3 [87]
Process Technology	90nm	90nm	65nm
Core Operating Voltage	1.2	1.2	1.2
Inputs per LUT	4	4	4
Embedded Multiplier Input Data Width	16	8	8
Embedded Ram Data Width	36	18	36
Number of Global Clock Networks	8	16	20

Table 3.1: Characteristics of the FPGA architectures used in this thesis

of circuit area and power consumption [83]. However, this is only the case if the design implemented on the FPGA uses the embedded blocks available. All the FPGAs used in this thesis only have embedded memories and embedded multipliers, other FPGAs have more sophisticated embedded blocks. For example the Xilinx Virtex-4 FX FPGA has multiply accumulate blocks instead of embedded multipliers, embedded memories, an embedded processor, and transceiver blocks to support high speed Input and Output. Table 3.1 summaries the features of the FPGAs used in this thesis.

3.2 FPGA Power Consumption

There are three types of power consumption in a FPGA, configuration power, static power and dynamic power [82]. For flash based FPGAs configuration power is not an issue because the configuration operation is not required. For SRAM devices, which are focused on in this thesis, the significance of configuration power to overall FPGA power consumption is dependent on how the FPGA is used. If dynamic reconfiguration is employed, or if the FPGA is regularly powered down, configuration power will obviously represent a greater proportion of overall FPGA power consumption. In most current designs, dynamic reconfiguration is not used. Consequently it is static and dynamic power which comprise the greatest proportion of power consumption in an FPGA.

As FPGA power consumption has become of greater concern there has been a significant amount of research into characterizing [82], predicting [88] and reducing both FPGA static and dynamic power consumption. The techniques for reducing FPGA power consumption proposed include, FPGA architectural modifications [89] with an associated place and route or synthesis algorithm if required, and place and route/synthesis algorithms independent of any FPGA architectural modifications.

3.2.1 FPGA Static Power

Static power consumption in an FPGA can be significant because, to support reconfigurability, an FPGA uses a large number of transistors. Each transistor will suffer from current leakage to some degree [90]. Moving to smaller process geometries increases the current leakage associated with each transistor. This has increased the significance of static power consumption and motivated the

development of techniques to reduce it.

In [90] an analysis of the static power consumption of the 90nm Spartan-3 FPGA is performed. It showed that 38% of the static power is consumed in the SRAM configuration cells. To reduce this two techniques have been proposed [89][91] and implemented in the more modern commercially available FPGAs. These are increasing the threshold voltage of the configuration transistors and increasing the gate oxide thickness of the configuration transistors. Each technique targets a different leakage type. Increasing the voltage threshold reduces the sub-threshold leakage. Increasing the gate oxide thickness reduces the gate leakage. Both techniques increase the time required for FPGA configuration, making the use of dynamic reconfiguration less appealing.

Further FPGA architecture modifications have been proposed to reduce the static power consumption in the FPGA interconnect and in the LUT multiplexer logic. According to the study in [90] these two components represent the second and third most significant sources of FPGA leakage current. As FPGAs are only available in discrete sizes, a design will only ever use a proportion of the FPGA resources available. However, the unused FPGA resources will still consume leakage power. By using a low leakage transistor to disconnect the unused resources from the power supply, power gating, the leakage current through the unused resources can be minimised [92] [89]. It also enables areas of the FPGA that are not used frequently to be disabled, in a similar manner to the use of power gating in ASICs [92]. An extension of this method is to use dual supply voltages. This allows FPGA resources to operate at a higher voltage if they are on a design's critical timing path and at a lower voltage if they are not. This potentially reduces both the static and dynamic power consumed by an FPGA [93].

Depending on the granularity with which power gating or dual supply voltage

techniques are implemented they can increase the FPGA area significantly. This is especially in the case of the dual supply technique as level converters are required to allow low voltage logic to drive high voltage logic without causing leakage current. FPGA performance is also reduced by the addition of gating transistors. Thus, the use of these techniques within an FPGA architecture needs to be carefully considered.

Methods to reduce the FPGA leakage which can function for any FPGA are proposed in [94]. Based on the principle that on average a LUT outputting a logic 1 will consume less power than a LUT outputting a logic 0, [94] proposes altering the configuration of LUTs to increase the probability of them outputting a logic 1. As this inverts the LUT output, only LUTs driving other LUTs can be subject to this modification in order that the inverted signal can be corrected at a later logic stage. Favouring the use of interconnect resources with a low leakage characteristic is also proposed in [94] to further reduce FPGA leakage power. When combined these methods result in a leakage power reduction of around 30% for a 90 nanometre FPGA. However, as the leakage characteristic of each interconnect resource may not necessarily be proportional to the interconnect's capacitance, favouring interconnect resources with a low leakage characteristic could potentially increase FPGA dynamic power consumption.

3.2.2 FPGA Dynamic Power

The design implemented on a FPGA has a greater effect on FPGA dynamic power consumption than on FPGA static power. Dynamic power consumption can be modeled using the equation,

$$P_{dynamic} = \sum \frac{1}{2} C_i V_i^2 \alpha_i \tag{3.1}$$

where C_i, V_i and α_i represent the capacitance, voltage and switching activity of resource i respectively. Equation 3.1 models the power consumed charging and discharging the capacitive load associated with each resource. This is the most significant source of dynamic power consumption. Additional dynamic power is consumed as a result of the short circuit created when an output transitions from *logic 0* to *logic 1* or vice versa. As this source of power consumption is also linearly dependent on switching activity, C_i in equation 3.1 can be increased to account for it. Thus in the extreme case if a design's switching activity, a function of both the design itself and its inputs, is negligible then the design's dynamic power consumption will also be negligible. However, this implies a design with negligible functionality.

Studies of FPGA dynamic power consumption have shown that 50% to 70% of dynamic power is consumed within the FPGA interconnect [82]. The large power consumed within the FPGA interconnect is due to each interconnect segment having a large capacitance associated with it. The large capacitance being due to the number of switch logic transistors which have to be connected to each interconnect segment to support reconfigurability. To reduce the dynamic power consumed within the interconnect a number of power-aware synthesis and place and route algorithms have been proposed. In [95], a technology mapping algorithm which attempts to consume high activity nets inside LUTs, thus removing the need for them to utilize any FPGA interconnect, is proposed. In [96], it is proposed to attempt to pack LUTs into clusters in a way which allows high activity nets to be completely routed within a single cluster. This reduces power consumption because interconnect local to a cluster typically has a much smaller capacitance than any of the global interconnect resources available on the FPGA. There are also similar algorithms, taking into account net switching activity, for

both placement and routing. Prioritising power within both the synthesis and place and route algorithms may conflict with a design's speed and area goals. For instance, nets which are on a design's critical path, which could be packed into a cluster, may not be as a result of prioritising the packing of nets with a high switching activity. As a result, the delay on a design's critical path may be increased.

All the power aware synthesis, and place and route algorithms proposed, require switching activity information in order to operate. To even derive an estimate of dynamic power consumption, switching activity information is required. The methods used to obtain switching activities are based on either design simulation or on probabilistic techniques. Simulation based methods offer a high degree of accuracy. However, as design switching activity is input dependent, it is difficult to find a set of inputs which are representative of real world operation and are also small enough for the simulations to be practicable. Probabilistic techniques generally offer an increase in speed, but sacrifice some accuracy. For example, by neglecting to account for correlations between signals present within the design.

Obtaining the switching activity information is further complicated by the issue of glitches. Glitches are spurious transitions which occur, on LUT or other FPGA logic outputs, as a result of unequal timing delays associated with the LUT or other FPGA logic inputs. In [88], a method of predicting the extent of additional switching activity due to glitching, prior to the place and route process, is proposed. The method proposed takes into account factors known at this stage, such as LUT logic function, combinatorial depth and logic path length, in order to determine whether glitches will be generated or propagated at each logic element. It is suggested in [88] that glitch prediction prior to place and

route is inherently difficult. A large number of glitches are a result of the variable delays introduced by the different interconnect resources used to route a design's signals. Consequently, essential information required for glitch prediction is not available prior to placement and routing.

Glitching, and hence FPGA dynamic power consumption, can by reduced through pipelining [97][98] and re-timing [99]. Both pipelining and re-timing reduce glitch propagation, preventing glitches at a LUT output from propagating onto the high capacitance interconnect, and hence the inputs of other LUTs in the system. Pipelining increases the design latency but, provided this can be tolerated, power consumption can be reduced by up to 80% depending on the design and the number of pipeline stages implemented [97]. It may be difficult, however, to know in advance what parts of the design will most benefit from pipelining. Time consuming modifications late in the design cycle may therefore be required in order to reduce power consumption. To prevent this, an automated method for including additional pipeline registers is proposed in [98]. The additional registers are clocked using a phase shifted versions of the non-pipelined register's clock. This ensures the additional registers do not affect the design's operation. The dynamic power reduction achievable with this method is less than that which can be achieved through pipelining early in the design cycle. The additional phase-shifted clocks must be distributed through the FPGA, creating an additional source of power consumption. In addition, the number of additional pipeline registers is restricted by the timing margin, or slack, available in the design. Nevertheless power reductions of up to 30% have been cited using this method [98].

The embedded blocks present in modern FPGAs can be taken advantage of to reduce dynamic power consumption [83]. As these blocks are effectively ASIC

circuits that implement a particular function, it is inherently more power efficient to use them for that function than it is to use the generic LUT fabric. However, if the dedicated blocks are not used in the design implemented on the FPGA they are effectively just additional sources of static power consumption. If the FPGA, or reconfigurable array, is targeting a specific application then the whole array, and each block in it, can be designed for that application. For example, in [100], a reconfigurable array targeting motion estimation is proposed. For commercial FPGAs the embedded blocks present must be flexible enough to be used throughout the application range the FPGA is targeting [101]. In commercial FPGAs therefore, embedded blocks are typically limited to RAM blocks, multiply or multiply accumulate blocks, clock control blocks and specialised input/output blocks to support high speed FPGA input and output. To reduce power consumption, the design implemented on the FPGA must be designed to use the embedded blocks available efficiently [83].

Techniques for minimizing the amount of power consumed in embedded RAMs are proposed in [102]. The techniques proposed take advantage of the fact that as a result of the way the synchronous RAMs in an FPGA operate, the majority of their dynamic power consumption is not dependent on the input data or input addresses but on the number of clock cycles the SRAM clock is enabled [103]. Two methods are proposed. One is to remap write/read enables to write/read clock enables. The other is to map logical memory blocks which are larger than the physical memory blocks available on the FPGA in a way which minimises the number of physical memories which have to be active for any read/write operation. Embedded RAM optimisations are important with respect to video encoder designs. Such designs generally make use of a large number of RAMs in order to reduce bus/memory bandwidth to an acceptable level.

The embedded RAMs on an FPGA can also be used to implement logic [104]. In terms of raw silicon area, implementing logic using embedded RAMs is more efficient than implementing logic using the FPGA LUT fabric [104]. From a power view point, implementing logic in the available embedded RAMs may reduce the size of the FPGA required for a design, and hence reduce the static power associated with it. However, it was shown in [105] that implementing logic in embedded RAMs results in an increase in FPGA dynamic power consumption. Thus, in general, using embedded RAMs to implement logic will result in an increase in a design's overall power consumption.

Clock gating techniques can also be used in FPGAs to reduce power. Two clock gating methods can potentially be used [106][107]. The first uses each FPGA register's enable pin to approximate the effect clock gating would have, the other uses the built-in FPGA clock management resources to actually disable one of the clocks present in the design. The second method reduces power consumption by the greater amount, as with the first method the clock is still being distributed to each register. The advantage of the first method is that it does not require a separate clock distribution tree, a finite FPGA resource, for each gated clock present in the design.

3.3 Summary

This chapter has introduced the basic architecture used by the majority of FPGAs available today. A key aspect of modern FPGA architectures is that they are heterogeneous. At a minimum a modern FPGA consists of LUTs, embedded RAMs and multipliers. The inclusion of additional blocks has been key to allowing more complex systems to be implemented on FPGAs.

Due to the number of transistors required by an FPGA, the static power consumed is inherently greater than that of an ASIC implementing an equivalent function. Significant research has been conducted and new features incorporated into modern FPGAs in attempt to reduce static power consumption. In the main these features are implemented at the FPGA architectural level. For example, the use of a different oxide thickness for configuration transistors.

There is more scope to influence dynamic power consumption at the application level because there is a direct link between the application implemented on the FPGA and the power consumed by it. That said, a number of synthesis, placement and routing algorithms, which aim to reduce dynamic power consumption, have been proposed. In chapter 5, results will be given showing how some of these techniques affect the power consumed by a FPGA based video encoder. The techniques proposed in [103] being used to reduce the power used by the embedded RAMs present in the FPGA video encoder design.

Chapter 4

FPGA Video Compression Systems

FPGAs have become a popular method of implementing real-time video encoding systems. Modern FPGAs are well suited to the video compression task. An abundance of logic (LUTs and embedded multiplier/DSP blocks) allows the complex algorithms required to be efficiently implemented. The embedded RAM present on all modern FPGAs allows the video data to be buffered on chip, reducing external memory bandwidth requirements.

A video encoder is only a single part of an overall system. In general, a video encoding system will include, a video input source, memory, processor(s) and an output path. Any encoder, regardless of architecture, will have memory bandwidth and FPGA resource requirements which have a significant impact on the overall system design. In this chapter, the integration of a pipelined video encoder into a complete system is studied. Two example systems are described, each targeting a different application. The first, in section 4.1, describes a low cost H.263 video streaming system for security applications. The second, in

section 4.2, describes a high performance H.264 video encoding system targeted at video conferencing applications.

4.1 H.263 Encoder System using the Xilinx Platform

4.1.1 Design of an Pipelined Encoder suitable for integration into Xilinx platform based systems

The encoder is based on the design described in [108]. A diagram of the encoder is shown in Figure 4.1. It consists of five computational modules, the full pixel motion estimator, the half pixel motion estimator, the transform and quantiser, the inverse transform and quantiser and the variable length encoder. Each module processes macroblock units of data and stores its output in the appropriate RAM buffer. Double buffering is used to allow the encoder to operate in a pipelined fashion, as shown in Figure 4.2. In this encoder, the number of clock cycles required per pipeline stage is fixed at 1468. To encode D1 (704x480) sized images at 30 frames per second therefore requires a minimum encoder clock frequency of approximately 59 megahertz (equation 2.3).

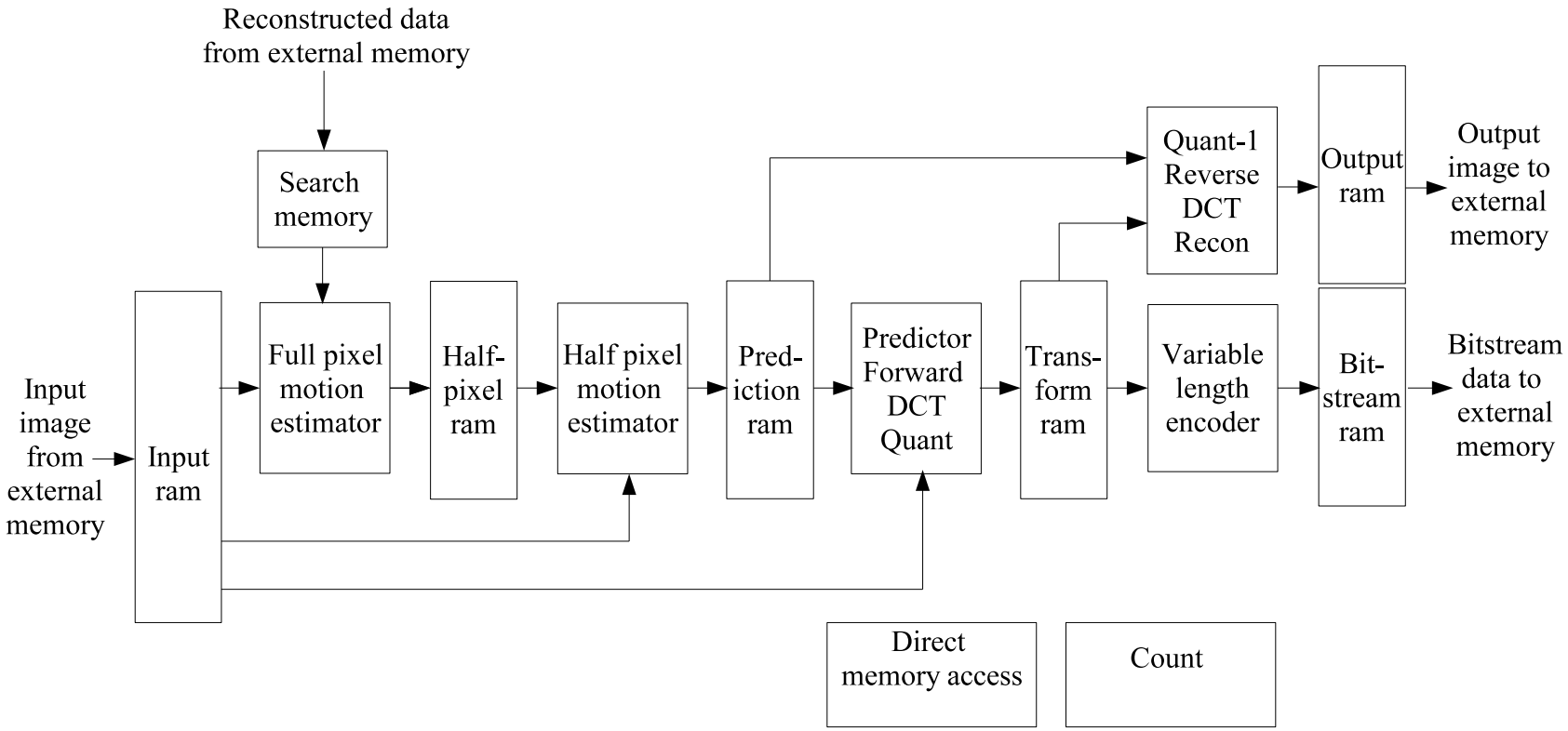

Figure 4.1: H.263 encoder architecture

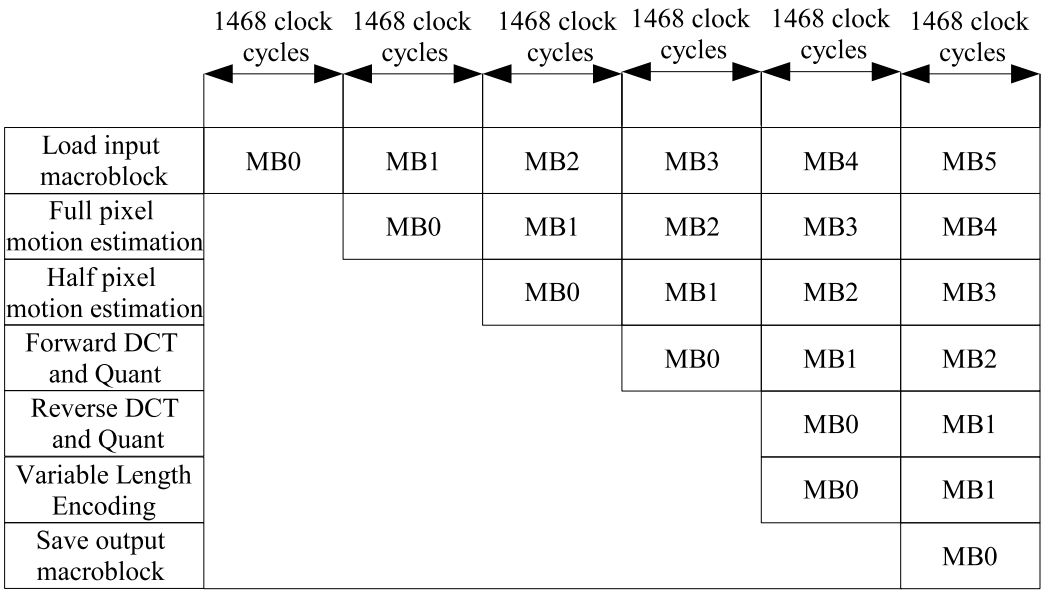

Figure 4.2: Pipeline operation of H.263 encoder. Each macroblock (MB) is processed sequentially through each of the encoder stages

Encoder Modifications

In order to ease integration into Xilinx platform based systems a number of changes were made to the encoder. The number of embedded RAMs used by it were reduced and a more efficient method of writing data to and from the encoder was introduced.

The encoder presented in [108] used 22 embedded RAMs for buffering the macroblock (MB) data between the computational modules, internal storage within the five functional modules, and buffering the reconstructed image data for use by the full pixel motion search module. This represents a significant proportion of the embedded RAMs present in mid-range Spartan-3 devices and, if not reduced, would have prevented the encoder being targeted at these FPGAs, given that in a typical Xilinx platform based design the microblaze processor also uses a number of block RAMs.

Two methods were used to reduce embedded RAM usage. Where possible, the

embedded RAMs were reduced by storing the macroblock data in a more compact fashion. For instance the search memory is required to store 12 macroblocks at any instance, the data being held, from a schematic point of view, in 4 buffers each storing 3 macroblocks. The previous implementation mapped each buffer directly to an embedded RAM. This, while simple, wastes more than half of each embedded RAM's storage capacity of 2048 bytes. To reduce the embedded RAMs used, the 4 buffers were re-mapped onto 3 embedded RAMs, the minimum number required to store 12 macroblocks of image data. The second method used was to utilise the distributed RAM feature available in Xilinx FPGAs, mapping some of the smaller memories required by the individual functional modules to distributed RAM.

The previous encoder stored all frames present in external memory (input frame, output frame and reconstructed frame) in raster scan format. The encoder operates on macroblock units of data. Thus, when loading and saving macroblock units of data the memory addresses the encoder accessed were not in a straight forward sequence. This is not an issue if static memory such as SRAM is used, as there is no performance penalty for addressing SRAM in a non-sequential fashion. However, when using dynamic memory, or when accessing a memory through a bus such as the On-chip Peripheral Bus (OPB) used here, it is desirable to use sequential memory addressing because the data can be transferred more efficiently through the use of the burst mode of the memory/bus. For the output and reconstructed frames the obvious method is to write out output frame in the macroblock ordered format shown in Figure 4.3. The appropriate data can then be easily loaded when compressing the next frame in the video sequence.

With respect to the input frame, the issue is more complicated due to it being coupled with the design of the camera interface. If the encoder accepts

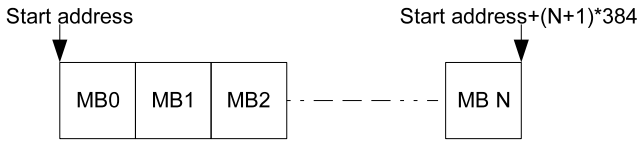

Figure 4.3: Macroblock ordered format used for reconstructed and output images

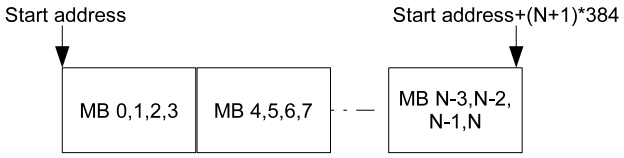

Figure 4.4: Macroblock ordered format used for input images

input frames in the format shown in Figure 4.3, the onus is put on the camera interface to write the input frame data to external memory in that format. Since any camera interface will receive camera data in raster scan order, its ability to write that data to memory efficiently is inhibited if it must write it out in the format shown in Figure 4.3. With the bt656 source used, the data must also be converted from an interlaced 4:2:2 format to a non-interlaced 4:2:0 format.

To do this and be able to write out the data in an sequential manner, the camera interface uses two embedded RAMs to double buffer one line of image data. For the first field it captures the chrominance and luminance data, for the second field it captures the luminance data only, thereby performing the 4:2:2 to 4:2:0 conversion required. Each image line is written out in 16 32-bit word bursts of luminance and chrominance data. This forces every 4 macroblocks of data to be intermingled as shown in Figure 4.4. However, it allows a burst size of 16, instead of 4, to be used to write the input frame data to external memory.

Encoder Interface

The camera interface and the encoder have internal registers which require to be set for every frame captured/encoded. To facilitate this within the Xilinx

platform systems being targeted, two OPB slave interfaces, implemented using the Xilinx provided OPB-IPIF module [109], were used. As already mentioned OPB masters interfaces were used to provide the encoder and camera with access to external memory. The external memory being accessed through any of the OPB external memory interfaces provided by Xilinx. This allows the encoder and the camera interface to be flexible with regard to the type of external memory used with them.

4.1.2 System Design

To demonstrate the use of the encoder within a low cost system the Avnet Spartan-3 1500 evaluation board [110] was used with two expansion cards, one to provide the bt656 input for the camera interface [111], and one to provide access to a 32-bit SDRAM memory [112]. The system implemented is designed to stream H.263 compressed video using the Real-time Transport Protocol (RTP).

The architecture of the Spartan-3 system is shown in Figure 4.5. The architecture used decouples as much as possible the encoding operation from the streaming operation. The encoding operation is centred on the encoder OPB bus, the streaming operation is centred on the processor OPB bus. The only data required to flow between the two buses is the H.263 bitstream data. This is written into the 8 kbyte embedded RAM buffer accessible to both buses, by the OPB DMA controller, under direction from the microblaze processor. At no point does the processor access the SDRAM directly. The encoder, camera interface and processor all operate concurrently, with the camera interface capturing image n, the encoder encoding image $n-1$, and the processor streaming image $n-2$.

This architecture was chosen for a number of reasons. Firstly, using two OPB

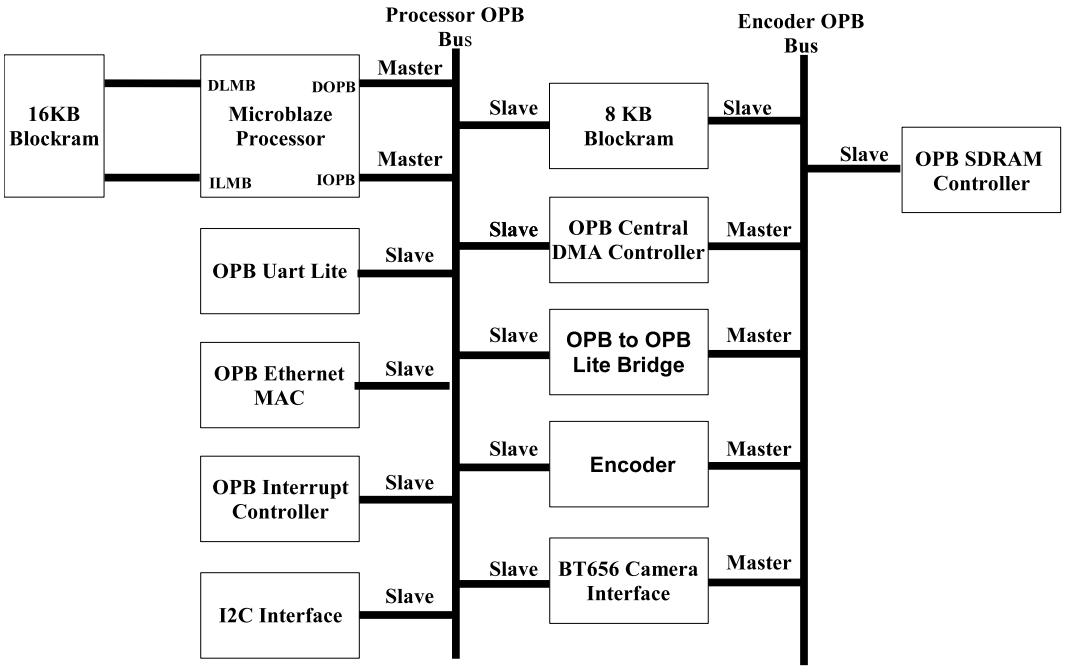

Figure 4.5: Spartan-3 System Architecture

bus segments ensures that access to the SDRAM is not impeded by the processor accessing the other slaves in the system. Secondly, the microblaze processor accesses the OPB bus using single read/write transactions, taking at least 10 clock cycles to read a 32-bit word from SDRAM, and 6 clock cycles to write a 32-bit word to SDRAM, due to the latency associated with the OPB bus and SDRAM interface [113]. Thus, if the processor software was to be run directly from SDRAM, it would be able to execute a maximum of 6 million instructions per second (assuming a system clock rate of 60 MHz), but realistically less, as processor access to the SDRAM will be hindered by the camera interface and encoder. Given this, it is clear that a degree of buffering of the program data will be required for the system to operate at the desired frame rate and resolution.

This buffering can take one of two forms, caching and directly storing the program instructions and data on the FPGA. Both methods use the embedded

RAMs present on the FPGA. In this system, 12 embedded RAMs are available to be used for this purpose. The others are used, in the encoder and camera modules as discussed in section 4.1.1 and in the ethernet media access controller, which requires a minimum of 4 embedded RAMs to function efficiently [114].

The advantage of using an instruction and data cache is that the SDRAM can still be used for storage. As a result the size of the system software does not become critical. However, the processor SDRAM usage when caches are used, is dependent on both the system software and the cache configuration. This is difficult to anticipate in advance and, for this reason, this approach was not pursued. Instead the 12 available embedded RAMs were used to store the system software directly. As shown in Figure 4.5 the available embedded RAM was divided with 8 2-kbyte embedded rams being connected to the microblaze processor through the faster local memory bus (LMB) and 4 being connected through the slower OPB bus. This division was chosen because the Xilinx supplied interfaces only support microblaze memory buffers which use a power of 2 number of embedded RAMs. It also allows the H.263 bitstream data to be directly loaded into memory accessible to the system software, as the second port of the OPB embedded RAMs can be accessed by the encoder OPB bus segment.

4.1.3 Software Design

The software controls the operation of the entire system, instructing the camera, and encoder, to capture and encode images, and transmitting the encoded images using the ethernet media access controller. Due to the system architecture currently used the total size of the system software is restricted to 24 kilobytes.

An Internet Protocol (IP) stack is required in order to properly format the encoded data for transmission over the ethernet media access controller. Although

it would be possible to develop a custom IP stack specifically for this system this would take a significant development effort, mitigating some of the benefit of using a platform design approach. There are a number of IP stack libraries compatible with the microblaze processor and ethernet media access controller. Xilinx supply the open source lwIP library and also a custom Xilnet library. There is also a port of the uclinux operating system available for the microblaze processor. Due to program size restrictions, the Xilnet library was used, as this is the only one capable of being implemented within the 24 kbytes available to store the system software. Although limited, the Xilnet IP stack is adequate for sending the UDP packets required by the RTP protocol. The Xilnet IP stack has very simple memory management. Two ethernet frame sized buffers are used, *sendbuf* for sending data and *recvbuf* for receiving data. These are mapped to the OPB embedded RAMs, with encoded data being loaded directly into *sendbuf* using the DMA controller shown in Figure 4.5.

Figure 4.6 shows a flow diagram of the software operation whilst encoding. Due to the program size restriction, no operating system is used. Instead, a camera and encoder generated interrupt triggers the software operation. The Xilnet library does not fragment IP packets across ethernet frames and, in any case, there is not the program memory available to support this. Thus, each RTP packet must be limited to approximately 1500 bytes. Given that the size of one H.263 encoded image is generally greater than 1500 bytes, this restricts the RTP payload header which can be used. The RFC 2190 payload header [115] cannot be used because, with this protocol, the H.263 bitstream data must be fragmented at image, group of blocks, or macroblock boundaries. With the current encoder design, the processor is only aware of where each encoded image starts in external memory, not where each group of blocks (GOB) or macroblock starts. Therefore,

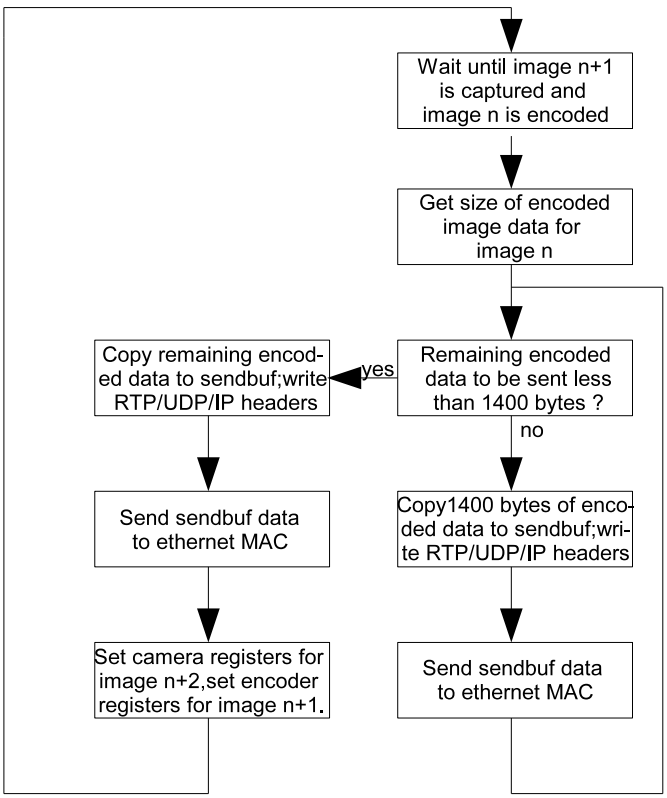

Figure 4.6: Flow diagram of software during encoding operation

the RFC 2429 payload header [116] was used as it allows each frame's H.263 bitstream to be fragmented at random points.

4.1.4 Results

Table 4.1 shows the FPGA resource usage of the revised encoder, discounting the additional de-interlacer component. Compared to the original encoder [108], the embedded RAM usage has been reduced significantly. The slice usage, however, has increase significantly compared to the 3000 slices used in the previous implementation. This is due to, the use of distributed RAM as opposed to embedded RAM, the addition of the OPB master and slave interfaces and the targeting of the encoder at lower cost Spartan series FPGAs as opposed to the more expensive

Slices	4813
Slice Registers	4364
LUTs	7061
Embedded RAMs	12
MULT18X18s	3
Maximum Encoder Clock Frequency	60 MHz

Table 4.1: Encoder resource usage (Spartan-3 1500 FPGA)

Slices	10944 out of 13312
Block RAMs	32 out of 32
MULT18X18s	6 out of 32
Maximum Clock Frequency	60 MHz

Table 4.2: Usage of Spartan-3 1500 resources by Streaming System

but higher performance Virtex series FPGAs.

The maximum clock rate the core is capable of operating at is just over 60 MHz in the lower speed grade Spartan-3 parts. In the higher speed grade parts it is capable of operating at over 69 MHz. Even using the lower speed grade parts, the encoder is capable of encoding D1 sized video at 30 frames per second. The minimum encoder clock frequency required for this throughput being 59 MHz.

Table 4.2 shows the resource usage for the overall system in the Spartan-3 1500 FPGA targeted. Using the lower speed grade part present on the Spartan-3 evaluation boards used, the system is capable of operating at 60 MHz and streaming D1 video at 30 frames per second. The size of the encoded data, however, is limited to approximately 1 MByte per second, due to the ethernet MAC only supporting 10 megabit operation at 60 mega-hertz.

The main limitation of the system is that, due to the software size constraint, the software has limited control functionality, only having a start streaming command which is sent to the system using the ethernet controller. Ideally, the system should be able to accept various commands, to start/stop streaming, send intra

frames and control the video bitrate. As the current system software already occupies 23506 bytes out of the 24576 bytes available alterations to the system hardware are required if this functionality is to be supported. One option is to use the Spartan-3E 1600 FPGA, which has an additional 4 embedded RAMs and is available for comparable cost to the Spartan-3 1500 FPGA.

Another option is to use caches and store the software within the external memory used by the encoder and camera interface. Sharing the external memory between the processor, camera interface and encoder becomes more feasible if the Xilinx cachelink interface is used. This custom interface was developed by Xilinx to provide the microblaze processor with a lower latency interface to external memory than that provided through the OPB bus. The caches used with the cachelink interface also supports larger cache line sizes than the caches used when external memory is accessed through the OPB bus. Use of cachelink interfaces to provide the encoder and camera interface with access to external memory would also provide a performance improvement whilst still allowing the encoder and camera interface to be easily used in a variety of Xilinx platform based systems with different external memories.

4.2 H.264 Encoder System using the Altera Platform

4.2.1 System Design

For this design, the maximum resolution required was 1280x720 (720p) at a frame rate of 30 frames per second. The target FPGA was the EP2C70 [84], the largest Cyclone 2 FPGA available. A modified version of the encoder described in [117]

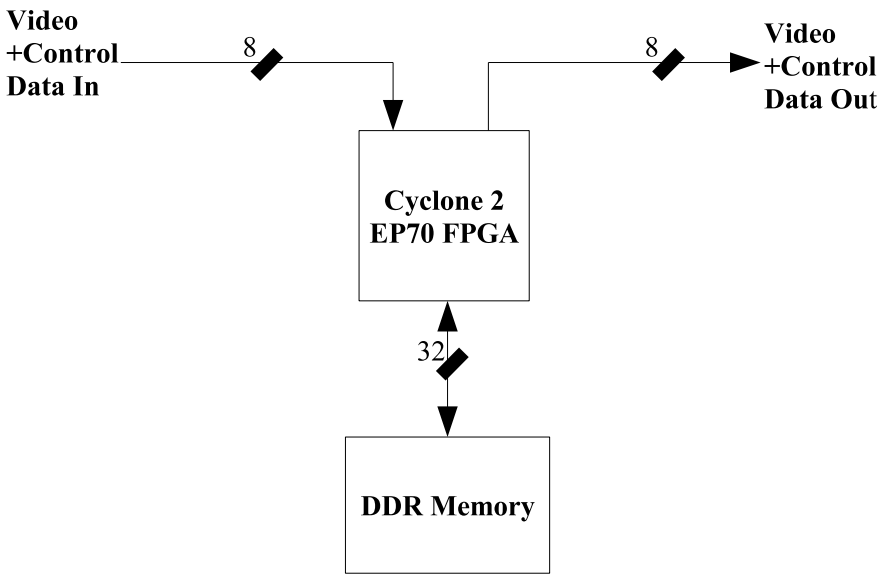

Figure 4.7: Overall system environment FPGA system operate in

was used. The FPGA encoding system differs from that discussed in section 4.1 in the method used for encoder control and in the method used to output bitstream data. The encoder in this system is controlled from a DSP device instead of an FPGA based processor. Encoding parameters being transmitted to the FPGA using the same bus that input frames are transmitted on. Bitstream data is outputted from the FPGA on a separate bus to another DSP device. The FPGA is not required to packetise the bitstream data for transmission on an IP based network. As a consequence of these differences there is no need for an FPGA based processor to support encoder operation in this system. A diagram summarising the overall system environment the FPGA design must operate in is shown in Figure 4.7. The DDR-2 memory shown in Figure 4.7 is used to store the reference frames required for motion estimation.

While the absence of an FPGA based processor simplifies the FPGA system design, this is offset by the encoding demands placed on the system. The encoder described in [117] uses a fixed 1400 clock cycles per pipeline stage. If only one

encoder instance was used a clock frequency in excess of 150 MHz would have been required to meet the desired frame rate and resolution (refer to equation 2.3). This would have been a difficult clock frequency to achieve in the low cost FPGA used. To reduce the clock frequency required, the slice based encoding technique described in section 2.2 was used. Two encoder instances being utilised to meet the desired frame rate and resolution.

Even when using two encoder instances a 75 MHz clock frequency would still have been required for an encoder C_p of 1400. This was a higher frequency than many of the encoder stages could operate at in the Cyclone-2 FPGA used. Using a third encoder instance was not an option. It would have required more resources than were available in the Cyclone-2 EPC70 FPGA. Therefore, the number of clock cycles that the full pixel motion estimation stage required was reduced. This allowed the C_p of the encoder to be reduced to 1200 clock cycles, reducing the required encoder clock frequency to 58.8 MHz. The actual encoder clock frequency used was 60 MHz. This allows both the encoder and input/output interface to use the same clock.

While the encoding process across can be split across multiple encoder instances, the memory transactions each encoder requires cannot be split across multiple external memories. Only one DDR-2 SDRAM was available for use on the target board. Even if more than one memory was available splitting the memory transactions would be non-trivial. A proportion of the reference frame data produced by each encoder instance must be accessible to the other. This allows the motion estimators in each encoder instance to consider search positions which cross the slice boundary.

Table 4.3 shows the maximum size of the main memory transfers required by the encoding system. Some of the smaller transfers required, such as the intra

Transfer Name	Megabytes per second
Reference Frame Load	124.4
Input Frame Load	41.5
Input Frame Save (From IO interface)	41.5
Output Frame Save	41.5
Loop Filter Load	51.875
Loop Filter Save	51.875
Total	352.35

Table 4.3: Maximum size of main memory transfers required by encoding system

prediction load, are not listed. Also not listed is the bitstream data which is saved to the DDR-2 SDRAM prior to being outputted on the data output bus. The size of this transfer is sequence specific. The maximum possible size is 1.75 megabytes per second, assuming the encoder complies with level 3.1 of the H.264 standard [1]. Achieving a memory utilisation of over 50% using the Altera DDR2-SDRAM controller used is difficult [118]. Given this, and the bandwidth required by the encoding system, it was suspected that it would be necessary to operate the DDR-2 SDRAM and its controller at the DDR-2 SDRAM's maximum clock frequency of 125 MHz. Simulations confirmed this suspicion.

The first architecture proposed for the encoding system is shown in Figure 4.8. In this architecture each encoder instance and the input/output module are separate Avalon bus masters. Therefore an arbiter is instantiated within the avalon switch fabric to control access to the DDR-2 SDRAM. The avalon switch fabric can also instantiate clock crossing logic when required. This was not used in this case however. The avalon switch fabric uses a slow handshaking protocol to correctly transfer data across asynchronous clock domains. This protocol is unsuitable for the large amount of data which needs to be transferred to and from the DDR-2 SDRAM. Instead dual-port embedded RAMs on the input/output of each encoder and within the I/O module are used to facilitate the efficient

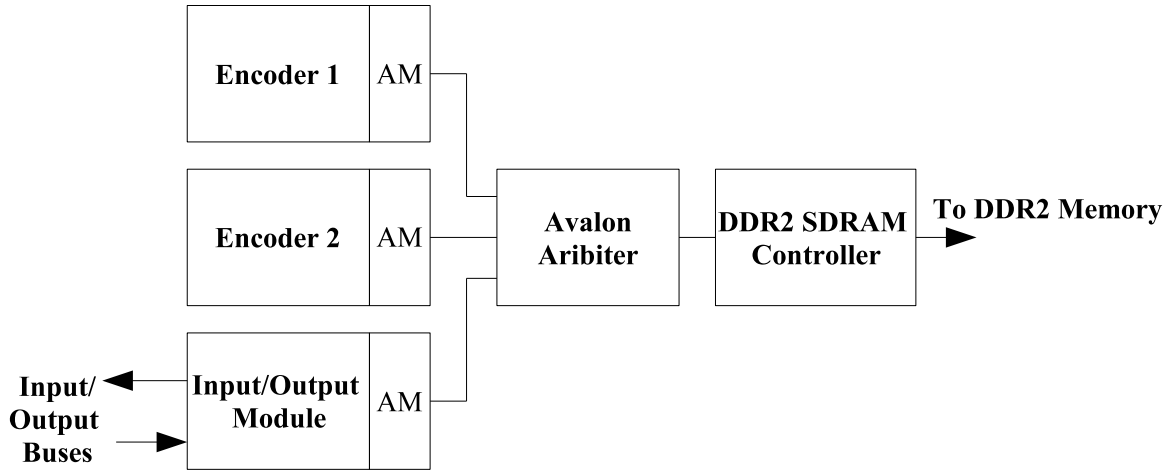

Figure 4.8: Proposed encoding system architecture

transfer of data between the core and memory clock domains. Consequently, the entire avalon switch fabric must be able to operate at 125 megahertz.

The main advantage of this architecture is its modularity. Treating each encoder instance and the input/output module as separate components makes it easy to reuse the encoder in other Altera platform based systems. For this specific system however, it was found to be impossible to use a modular architecture and still meet the system's frame rate and resolution requirements. This was due to the avalon switch fabric being incapable of arbitrating between multiple masters at 125 mega-hertz when implemented in a Cyclone-2 FPGA. To resolve this issue the two encoder instances and the input/output module were merged into the one avalon component. The three components sharing the one avalon master interface. This removes the need for an arbiter within the avalon switch fabric. Instead, an arbitration between the various modules takes place internally within the avalon component as shown in Figure 4.9. This allows the majority of the arbitration process to operate at the encoder and IO clock frequency of 60 MHz. Consequently the 125 MHz memory clock frequency required can be achieved

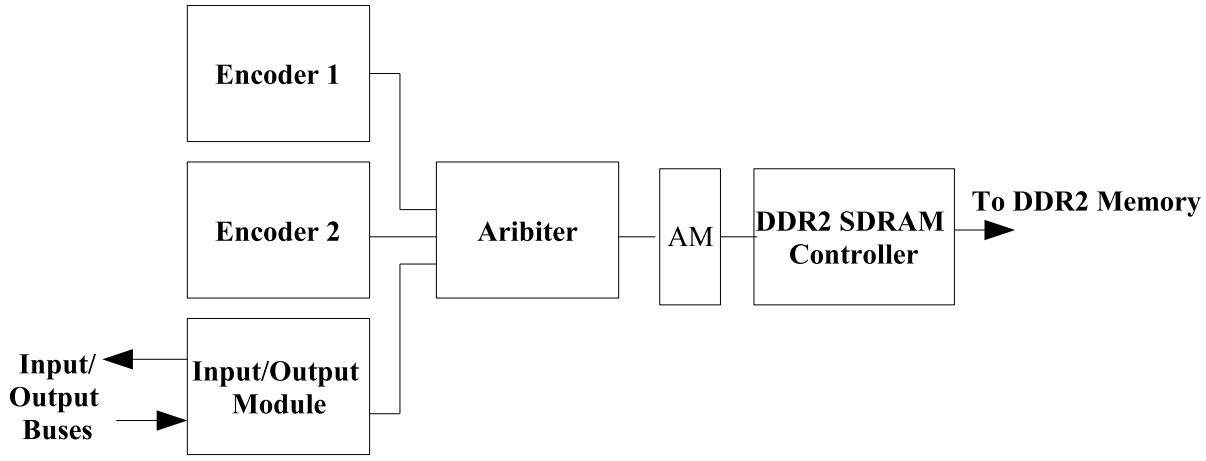

Figure 4.9: Actual encoding system architecture

whilst using an avalon bus interface.

4.2.2 Input/Output Interface

Format and Timing

The input/output interface must parse the data transmitted on the input bus, load the appropriate frames into the DDR-2 memory for use by the two encoder instances, and output the bitstream data on the output bus. In addition, it must be able to pass on the frames to be encoded to the output bus, as well as any additional frames transmitted on the input bus.

Data is transmitted to, and from, the FPGA in meta-frames. The meta frame format is used to ensure the FPGA is compatible with the other devices in the system. Each meta-frame consists of 625 lines of 3200 bytes. An 8 byte horizontal synchronisation period occurs at the end of each line. A 5 line vertical synchronisation period occurs at the end of each frame. The first 128 bytes of each input meta-frame contains the information required for encoder control. This is then followed by up-to 3 video frames, the first of which is the frame to

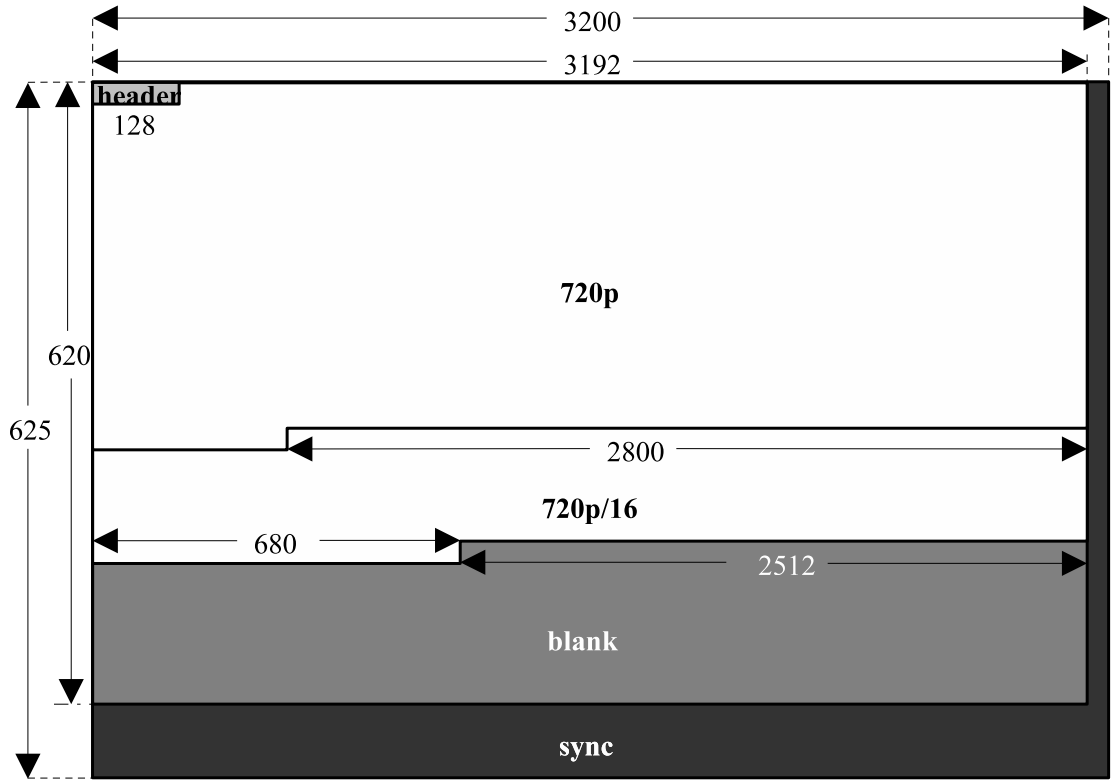

Figure 4.10: Example input meta frame

be encoded. An example input meta-frame is shown in Figure 4.10.

In the example shown in Figure 4.10, the 720p frame must also be passed on to the output bus as well as being encoded. Whether the frame to be encoded must also be passed on is dependent on the combinations of frames contained within the meta-frame. A complete list of the input frame combinations which had to be supported is given in table 4.4.

The requirement to pass on frames complicates the output meta-frame format. There are no spare resources available to store the additional pass-on frames in the FPGA. The DDR-2 memory cannot be used as there is no spare memory bandwidth available. Therefore, the frames must be passed on with minimal delay. As a consequence the bit-stream data must be transmitted in the empty

Encoded Frame Size	Encoded Frame Passed On ?	Pass On Frame Sizes
720p	Yes	720p/16
w4cif	No	720p
w4cif	Yes	w4cif/16
w3cif	No	720p
wcif	No	720p,720p/16
4cif	Yes	qcif
3cif	No	4cif,qcif
cif	No	4cif,qcif
4sif	Yes	qsif
3sif	No	4sif,qsif
sif	No	4sif,qsif

Table 4.4: Input frame combinations supported

space available within each output meta frame. For meta-frames which require the encoded frame to be passed on, the bitstream is located at the end of each meta-frame, as indicated in Figure 4.11. For meta frames which do not require the encoded frame to be passed on, the bitstream data starts at the end of a meta-frame and, if necessary, continues at the start of the next meta-frame as indicated in Figure 4.12.

From Figure 4.11, it can seen that the encoding latency with this type of meta-frame is over 30 milliseconds. The bitstream is transmitted on the output bus over 30 milliseconds after the corresponding input frame has been transmitted. This is unavoidable given the time needed to encode larger frame sizes and the inability to store the pass on frames. The latency for meta-frames where the frame to be encoded is not passed on is much less, as shown in Figure 4.12. This is a result of the smaller frame sizes which have to be encoded and the larger frame sizes used for pass on frames in this case. As a result the bitstream can be outputted in the same meta-frame as the corresponding input frame.

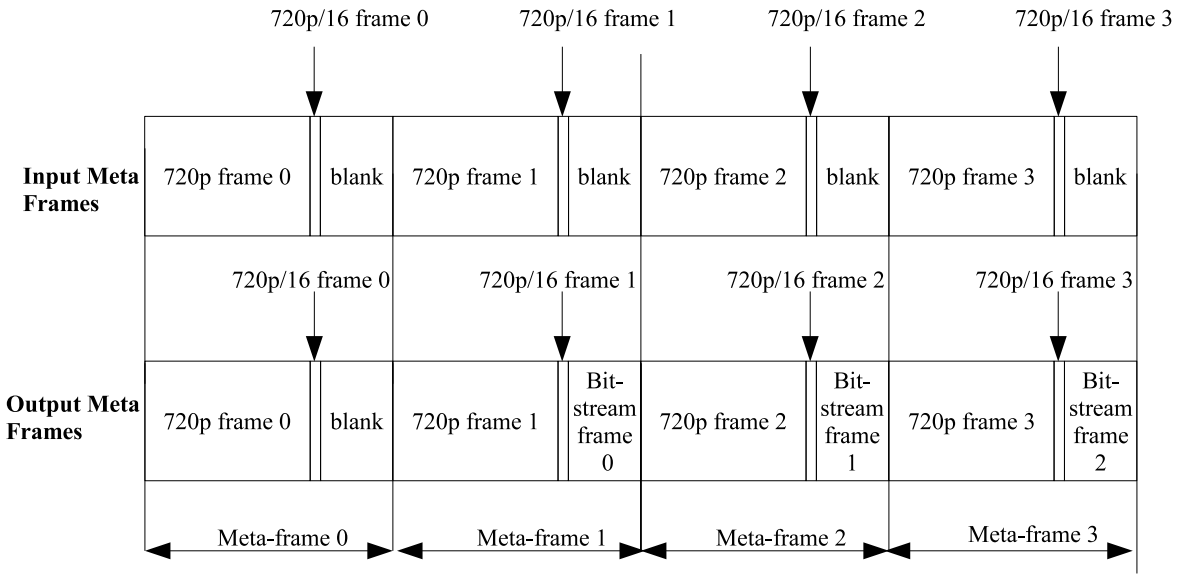

Figure 4.11: Timing relationships between the input and output meta frames when the encoded frame is passed on to the output bus

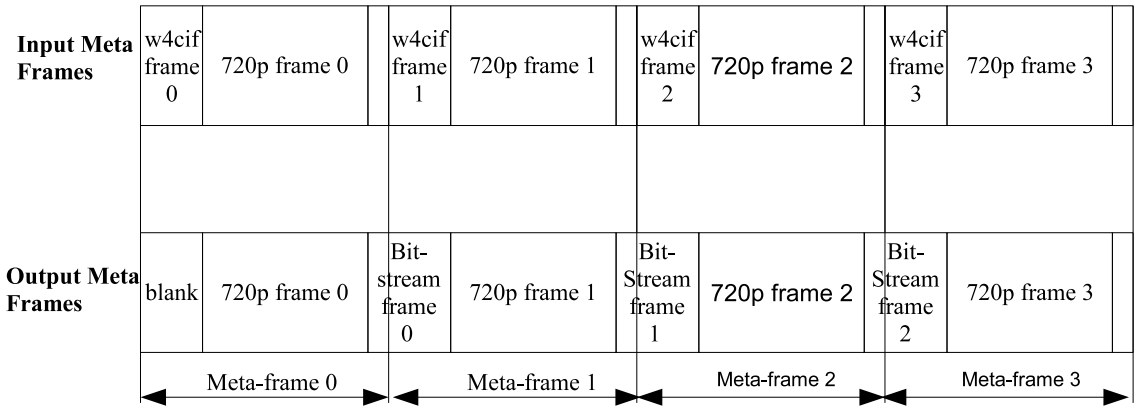

Figure 4.12: Timing relationships between the input and output meta frames when the encoded frame is not passed on to the output bus

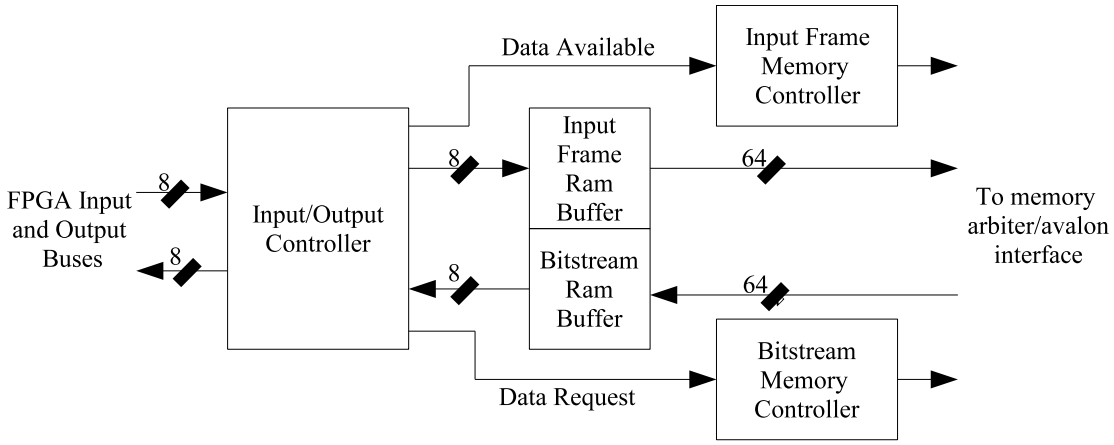

Figure 4.13: Structure of I/O interface

Input/Output Interface Design

The structure of the input/output interface is shown in Figure 4.13. It consists of three main components. The input/output controller communicates with the two external buses. It parses the header data from the input bus, writes frame data to the input frame memory buffer, and reads the bitstream data from the bitstream memory buffer and puts it on the output bus. The input frame memory controller schedules the transfer of the input frame from the input memory buffer to the external memory. The bitstream memory controller schedules the transfer of the bitstream data to the bitstream memory buffer. The memory buffers are used to transfer the bitstream and input frame data efficiently between the capture and memory clock domains.

Similarly to the design described in section 4.1, input images are received in raster scan order. For this design, the input image memory format shown in Figure 4.4 is not used. Instead, the simple macroblock ordered format shown in Figure 4.3 was used. The format shown in Figure 4.4 was not used because the Altera DDR-2 controller used only supports a burst length of 4. If an Avalon

Logic Elements	63929 out of 68416
Embedded Rams (M4K)	240 out of 250
Maximum Memory Clock Frequency	127.8 MHz
Maximum Encoder Clock Frequency	66.77 MHz

Table 4.5: Usage of Cyclone-2 resources and maximum operating frequency

master requires a larger burst size than that supported by the DDR-2 controller, the Avalon switch fabric instantiates additional logic to support it. It was found that this additional logic could not operate at 125 megahertz. Consequently, there would have been no benefit in using the memory format shown in Figure 4.4 in this system.

4.2.3 Results

The total resources used by the system are shown in table 4.5. From it it can be seen that, even with the architecture used, the memory clock frequency requirement is met with a minimal timing margin. Similarly to the system presented in section 4.1 the most heavily used resource in the design is the embedded RAMs. Although in this system the LUT utilisation is only slightly less than the embedded RAM utilisation.

4.3 Summary

The implementation of two FPGA video encoding systems has been described. Both systems used readily available intellectual property provided by the FPGA manufacturers to provide the video encoder with access to an external memory. In both cases this proved problematic. The effective bandwidth provided by the IP in each case only being suitable due to custom modifications to the encoding system. In the case of the system presented in section 4.1, a custom memory format was

used for input frames. In the case of the system presented in section 4.2 the overall system was specifically designed around the memory/bus architecture's limitations. This is not a surprising result, given the significant amount of data an encoder needs to write to, and read from, external memory.

Another common issue in the two systems is the high utilisation of FPGA embedded rams. This is unfortunate. The easiest method to reduce an encoder's memory bandwidth requirements is to store more data in the embedded RAMs available on the FPGA. The results presented here suggest that in many cases this will increase the size of the FPGA required to implement the encoding function. Thus, making the integration of the encoder into a complete system easier will, in many cases, increase the size of the FPGA required to implement the system.

Chapter 5

FPGA H.264 Video Encoder Power Analysis

In this chapter, the power consumed in an FPGA H.264 implementation is analysed. It is shown that, with respect to dynamic power consumption, the motion estimation function consumes the most power even when variable block sizes are not supported. Given the large computation required by the motion estimation function this is not an unexpected result. The analysis described in this chapter was performed to obtain real power consumption values on a commercial FPGA, to assess the extent which each video compression function contributes to overall dynamic power consumption and to determine the best methods to reduce the dynamic power consumed by 4i2i's H.264 encoder. 4i2i's H.264 encoder is studied, instead of the H.263 encoder used in the previous chapter, to ensure the result are relevant, from both a technological and business perspective.

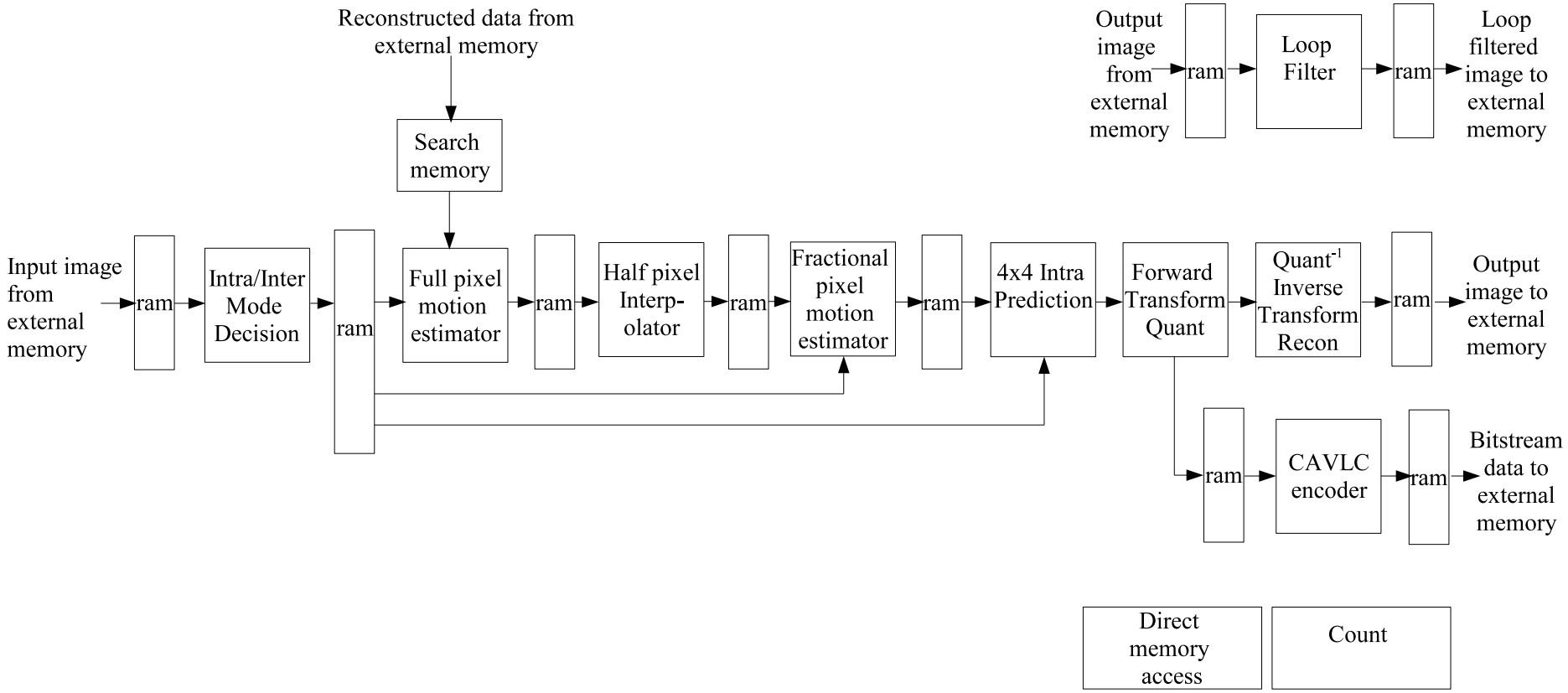

Figure 5.1: Simplified diagram of H.264 encoder studied. Note that the memories internal to each functional unit have been omitted from the diagram

5.1 Power Estimation Method

To estimate the power consumed by the encoder when implemented on an FPGA, the Altera Quartus-2 software was used. The Quartus-2 software was also used to synthesize, place and route, the encoder design on the Cyclone-2 EP70 FPGA targeted. The encoder studied uses an external memory to store the reference frames required for motion estimation and to store other information required during the encoding of a video frame. In common with all encoder implementations, the encoder studied caches search area data on chip allowing it to be re-used during the motion estimation stage of adjacent macroblocks in the video frame. Despite this, the encoder's external memory bandwidth requirements are still significant. It was therefore considered necessary to estimate the power consumed by the external memory in addition to the power consumed by the FPGA based video encoder. To do this a Micron SDRAM power model was used. The complete power estimation method is illustrated in figure 5.2.

In common with any encoder implementation, the frame rates and resolutions supported by the encoder studied are dependent on the supplied clock frequency and the available external memory bandwidth. As the encoder's clock frequency is increased its power consumption will also increase. In addition a higher clock frequency requirement gives the synthesis and place and route tools less scope to reduce power consumption. In this analysis, the aim is to derive the best possible power consumption for each frame rate and resolution being considered. Therefore, the encoder clock frequency was set to the minimum frequency required to support the frame rate and resolution being considered. For the pipelined encoder used, the required frequency for each frame rate and resolution can be derived using equation 2.3. C_p for the encoder studied is 1400 clock cycles.

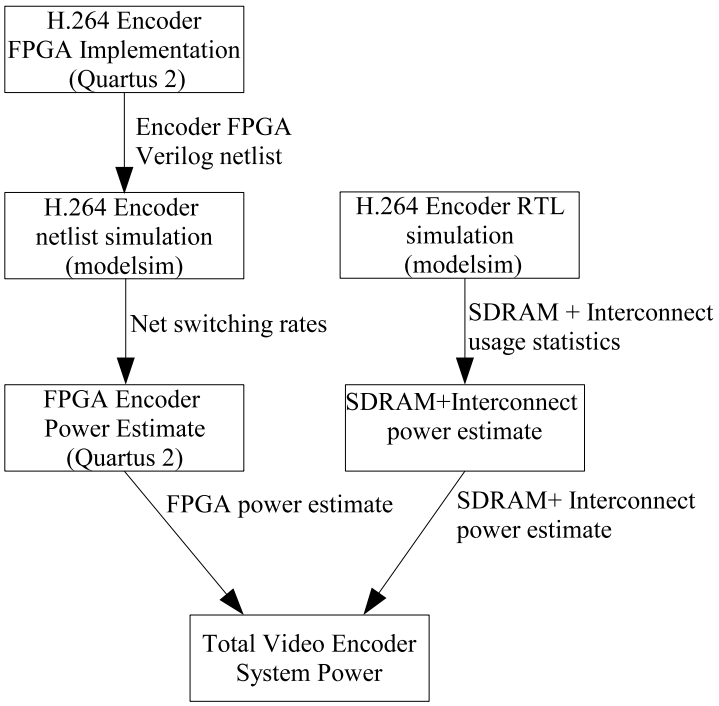

Figure 5.2: Method used to estimate power used by H.264 encoding system

While the encoder clock frequency is solely determined by the frame rate and resolution required, external memory usage, and hence the external memory's power consumption, is dependent on the design of the overall system. However, to simplify the power analysis, only the encoder's interaction with the external memory has been considered in this instance. Thus, the power estimates derived for the external memory and interconnect will only reflect the power consumed as a result of encoder operation. In a practical system the power consumed will be greater. At a minimum, additional power will be consumed writing input frames to external memory and reading the H.264 bitstream data from it.

A 32-bit SDRAM is assumed [119]. In order to provide the encoder with sufficient memory bandwidth, it was necessary to clock the memory at a higher frequency than the encoder. Simulations were used to determine the precise memory clock frequency required for each frame rate and resolution. This is

Sequence	Resolution	rame Rate (fps) F	Encoder Frequency (MHz) (MHz)	Memory Frequency (Mhz)
Football	352x240	30	17	40
Stefan	352x240	30	17	40
Mobile	352x288	30	17	40
Foreman	352x288	30	17	40
Garden	704x480	30	55	125

Table 5.1: Sequences, frame rates and clock frequencies used

not an uncommon situation. As shown in section 4.2, the minimum memory and encoder clock frequencies required will typically not be equal. As all input/output data from the encoder is from embedded RAM, it is relatively simple to switch clock domains due to the dual port nature of the embedded RAMs on modern FPGAs. The sequences, frame rates, encoder and memory clock frequencies used are shown in table 5.1.

5.1.1 Obtaining FPGA Switching Activity Information

As discussed in section 3.2.2, obtaining representative switching activities for an FPGA design is a non-trivial task. The Quartus-2 software supports a deterministic method for estimating switching activity. A Verilog netlist and associated TCL script are produced by the Quartus-2 netlist writer. The netlist, with an appropriate testbench, is then simulated using the Modelsim simulator. The TCL script produced instructs Modelsim to log all transitions which occur on nets in the design in a Value Change Dump (VCD) file. This is then parsed by the Quartus-2 power analyser to obtain estimates for net switching activity, and subsequently a FPGA power consumption estimate. This method is illustrated in Figure 5.3

When simulating a large design the size of VCD file used to log signal transi-

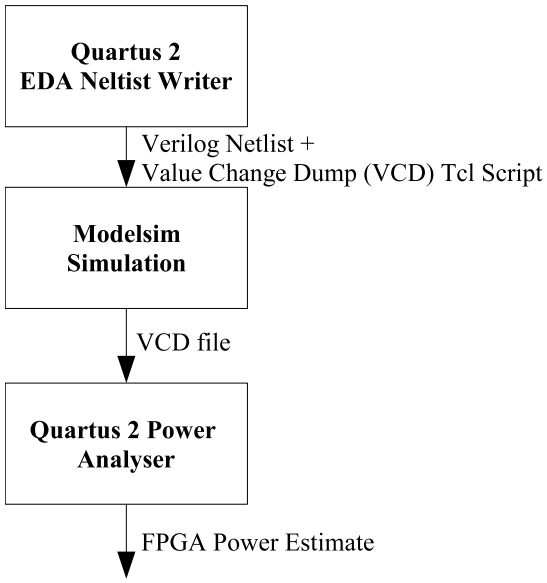

Figure 5.3: Power estimation flow directly supported by the Quartus-2 toolset

tions can become very large. This made it difficult to simulate the encoder for the length of time required to obtain realistic switching activity estimates. VCD file sizes in excess of one gigabyte were observed when simulating the encoding of just one frame of a video sequence. More importantly, it was found that the Quartus-2 power analyser would not read the large VCD files produced reliably. Therefore, an alternative method was developed to obtain switching activity estimates.

A diagram showing the alternative flow used is shown in Figure 5.4. VCD files are not used. Instead net transitions are recorded using the power information tracking functionality provided by Modelsim [120]. The TCL file produced by the Quartus EDA netlist writer is modified. The modified TCL file instructs Modelsim to track the appropriate nets for power information instead of logging the net transitions within a VCD file. At the end of simulation period Modelsim outputs the power information in a power report file, an example of which is shown in figure 5.5.

The power report file provides the number of transitions which occur on each

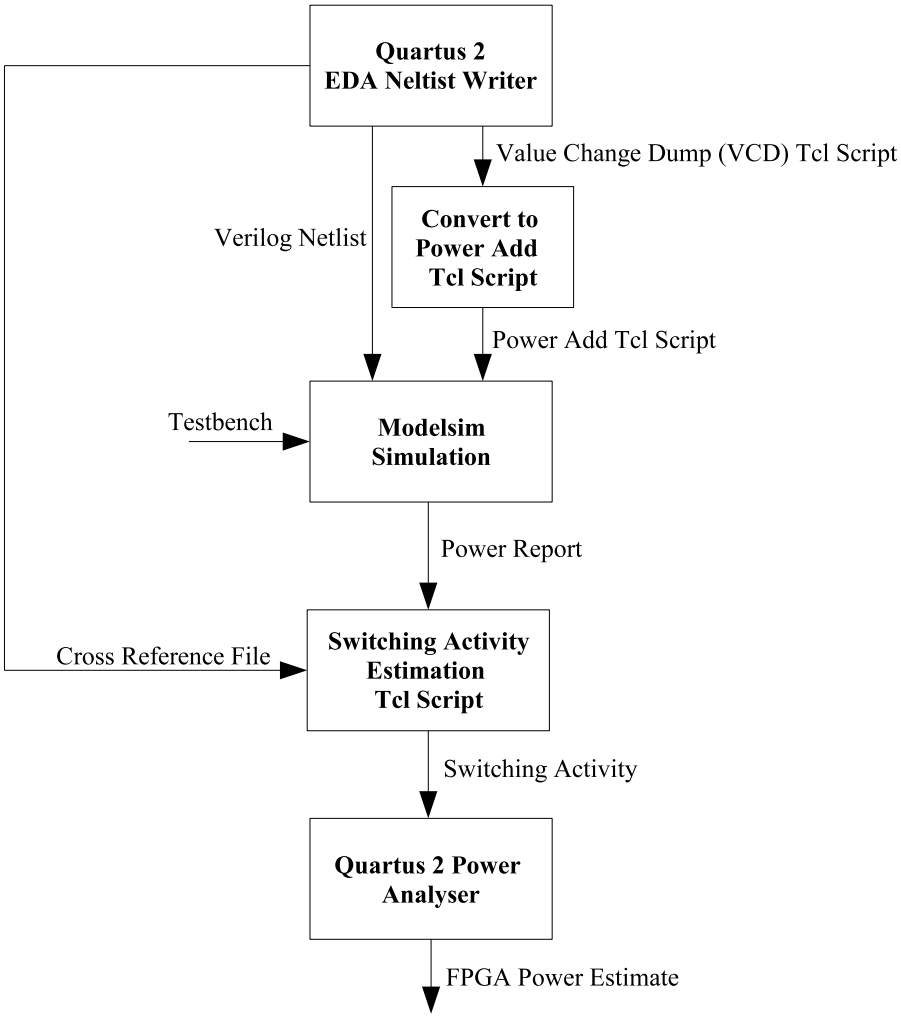

Figure 5.4: Modified power estimation flow used

```
362670000
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_actrl_aramip_rd_addr(2) 116821 0 7125455890 4237214110 0
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_actrl_aramip_rd_addr(3) 86461 0 6769055890 4593614110 0
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_ctrl_aramip_addr_int(2) 69337 0 8018095897
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_ctrl_aramip_addr_int(3) 51713 0 6977855897
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(0) 26442 0 5471120000 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(1) 26052 0 5509340000 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(2) 24807 0 5585115900 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(3) 22475 0 5702575895 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(4) 17700 0 5503840000 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(5) 9100 0 8986140000 2
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(6) 5172 0 9213500000 2
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(7) 3414 0 1538020000 9
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(8) 26332 0 5599440000 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(9) 25580 0 5712360000 5
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(10) 24641 0 5689335910
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(11) 22087 0 5701255910
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(12) 17230 0 5504520000
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(13) 8788 0 8718480000 2
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(14) 4978 0 9109780000 2
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(15) 3278 0 1611540000 9
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(16) 26590 0 5474320000
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(17) 26058 0 5290360000
otion_estimation_test/dut/the_motion_estimator/the_fractional_est_athe_quart_pel_est_arampre_wr_data(18) 24631 0 5641755906
```

Figure 5.5: Example power report produced by Modelsim

net. It also provides the time each net is at *logic 1* and *logic 0*. This enables a better estimate of the static power consumed. More importantly it ensures that the power estimate of FPGA blocks, such as embedded RAMs, whose dynamic power is predominantly determined by how often they are enabled is reasonably accurate.

The net names assigned to equivalent nets in Quartus-2 and Modelsim differ. The cross reference file is used to determine the equivalent Quartus-2 net name for a given Modelsim net name. This ensures that the switching activity and static probability information derived from the power report file is assigned to the correct nets prior to the execution the Quartus-2 power analyser.

When a VCD file is used to log transitions the time each transition occurs is stored. This allows the glitch filtering algorithm embedded in the Quartus-2 power analyser to discount transitions which do not result in a full transition on the output logic and interconnect. Specific details on the glitch filtering algorithm used are not available [121]

No timing information is present in the power report file. It is therefore impossible, when the modified estimation method is used, to discount the appropriate transitions after the simulation has finished. This is a problem. If no glitch filtering algorithm is applied the power used by the encoder will be significantly over estimated. To solve this problem a crude glitch filtering algorithm is applied. Instead of using a transport delay model for the netlist simulation, as recommended in the Quartus-2 power analyser manual [121], an inertial delay model was used instead. This underestimates the power consumed by the encoder. The power consumed by glitches which are shorter than the delay associated with a resource output will not be modeled. The power consumed by all other transitions, including a glitch which lasts longer that delay associated with a resource output,

will still be taken into account.

5.1.2 SDRAM and Interconnect Power Modeling

The interconnect between the SDRAM and the FPGA was modeled using a simple lumped capacitance model, with the power consumed in the interconnect being calculated using equation (3.1). The capacitance C was taken to be 15 pf. This was based on I/O capacitance figures from the FPGA and SDRAM datasheets, and the assumption of a 3pF capacitance per PCB trace. The voltage V was set to 3.3V in order to be consistent with the SDRAM used and the number of transitions T was obtained from an RTL simulation as indicated in figure 5.2. The power consumed by the SDRAM itself was estimated using a model supplied by Micron [122]. The usage statistics it requires were obtained from RTL simulations of the encoder and SDRAM model.

5.2 Results

5.2.1 Validation of Power Estimation Method

To test whether the modified method for estimating FPGA power consumption gives result comparable with the standard Quartus method a test design was used. The test design used was the fractional pixel estimator described in section 7.4. It contains a mixture of embedded RAMs, multipliers and logic, providing a reasonable test of the accuracy of the modified power estimation method. The standard Quartus method and the modified method were used to estimate power of the same Cyclone-2 fractional estimator build. The *suzie* QCIF sequence was used as input to the test design. A sequence with a larger resolution was not

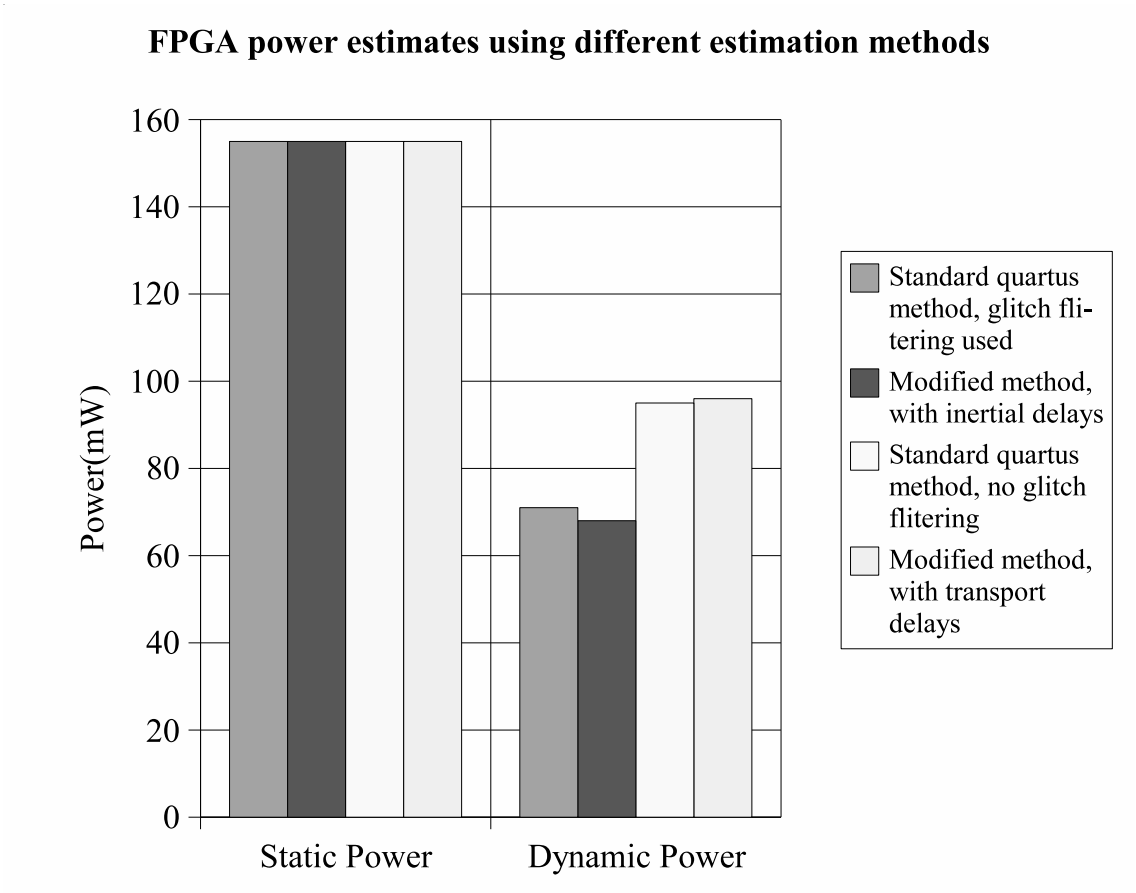

Figure 5.6: Comparison of static and dynamic power used using different estimation methods

used due to the Quartus-2 power analyser's inability to process large VCD files reliably.

Figure 5.6 compares the overall dynamic and static power consumption figures for the test design for four test cases, the standard Quartus method without glitch filtering, the standard Quartus method with glitch filtering, the modified method using transport delays and the modified method using inertial delays. From Figure 5.6, it can be seen that the static power estimate is not dependent on the estimation method used. This is to be expected given that glitch filtering will not significantly effect the time a LUT output is at *logic 1* or *logic 0*.

The dynamic power estimates do vary significantly. In the two cases where no glitch filtering is used, the dynamic power estimate is 30% to 40% greater than the two cases where a glitch filtering algorithm is employed. Given that a large proportion of this power will not be consumed in practice, using a glitch filtering algorithm is clearly justified. The modified method, when glitch filtering is used, gives a slightly lower power estimate than the standard Quartus method. This can be attributed to the different glitch filtering algorithms used.

Altera claims that that standard Quartus power estimation method, with glitch filtering enabled, has an error of +/- 20% when compared to "real world" power measurements [123]. The results given show that the modified method used produces a slighly lower power estimate than the standard Quartus method for the test design analysed. The difference between the two methods could potentially be greater, particularly if a design contains a large proportion of circuit elements which are prone to glitching. However, the majority of of encoder sections analysed here are similar to the test design analysed. Thus, the results presented should be of a similar degree of accuracy to those provided by the Quartus tool.

5.2.2 Overall Results

The total power figures for each sequence are shown in table 5.2. As would be expected, there is a marked increase in power consumed as the size of the encoded sequence is increased. The power consumption distribution, between the FPGA, SDRAM, and interconnect, for each sequence is shown in Figure 5.7. From Figure 5.7 it can be seen that when encoding small sized sequences the power consumed in the FPGA dominates, consuming over 70% of the total system power. In contrast, for the larger *garden* sequence, the FPGA consumes less

Sequence	Quantisation	Power Consumed (mW)				
		SDRAM	IO	FPGA Static	FPGA Dynamic	Total
Football	6	86	18	225	78	407
	20	86	17	225	77	404
	30	86	15	225	74	400
Stefan	6	86	16	225	76	402
	20	86	15	225	73	399
	30	86	14	225	73	398
Mobile	6	98	20	225	94	436
	20	98	19	225	91	432
	30	98	18	225	90	430
Foreman	6	98	21	225	89	433
	20	98	20	225	88	431
	30	98	12	225	85	427
Garden	6	312	70	225	297	904
	20	312	67	225	292	896
	30	312	63	225	287	887

Table 5.2: Power consumed encoding each sequence

then 60% of the total system power. This difference is a result of the high amount of static power consumed by the FPGA, approximately 225 mW. As the size of the sequence being encoded increases, the FPGA static power becomes less significant as the dynamic power consumed in the FPGA, SDRAM, and interconnect increases.

The FPGA static power for each sequence was estimated at a temperature of 25 °C. There were only minor variations between the static power consumed encoding each sequence. If the estimation process is repeated at 85 °C, the limit of the FPGAs temperature operating range, the FPGA static power is much greater, approximately 335 mW. It is clear from these results that FPGA static power cannot be ignored. However, the only direct way of reducing it at the application level is to reduce the size of the FPGA the application, in this case video encoding, requires. That said, reducing the dynamic power consumed in

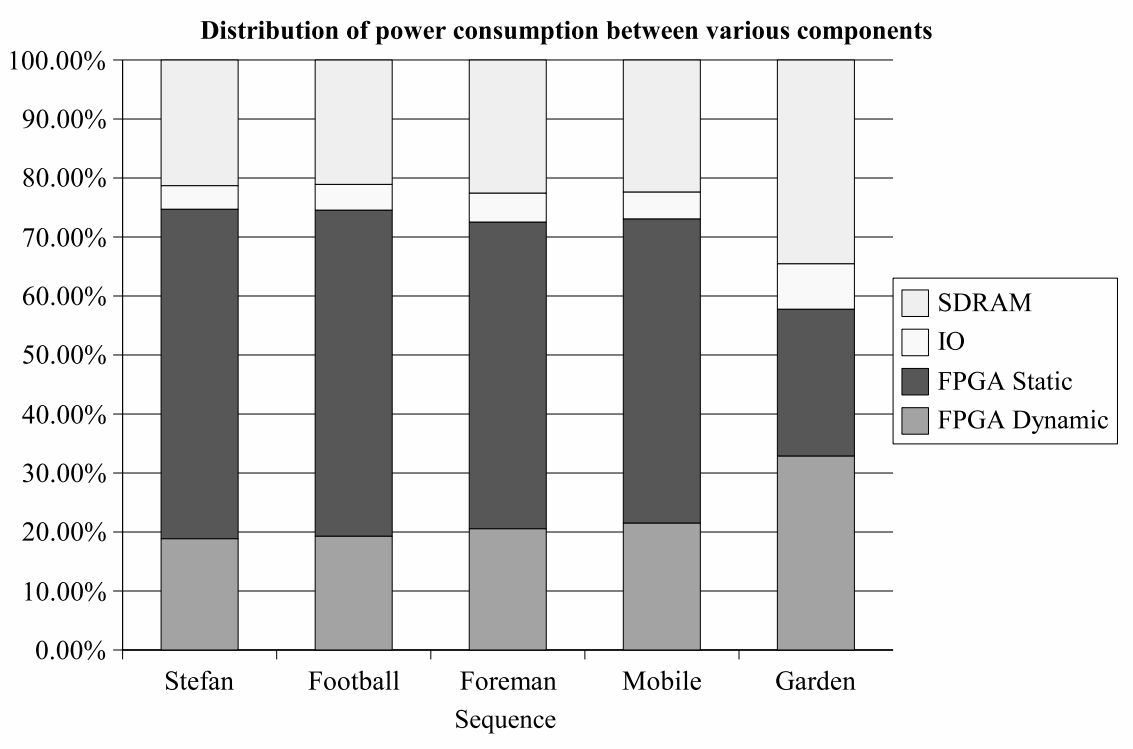

Figure 5.7: Overall power consumption distribution for each sequence

the FPGA will indirectly affect the static power consumed by the FPGA, as it will reduce the temperature the FPGA operates at in a particular environment.

5.2.3 FPGA Power Consumption

Power By Encoder Function

The dynamic power consumed in the FPGA by each encoder function is shown in Figure 5.8. A description of the various functions is given in table 5.3. The results shown in Figure 5.8 are when a quantisation parameter of 6 was used. There was negligible variation in the proportion of power consumed by each function when different quantisation parameters were used. Complete power results per encoder function are given in Appendix D

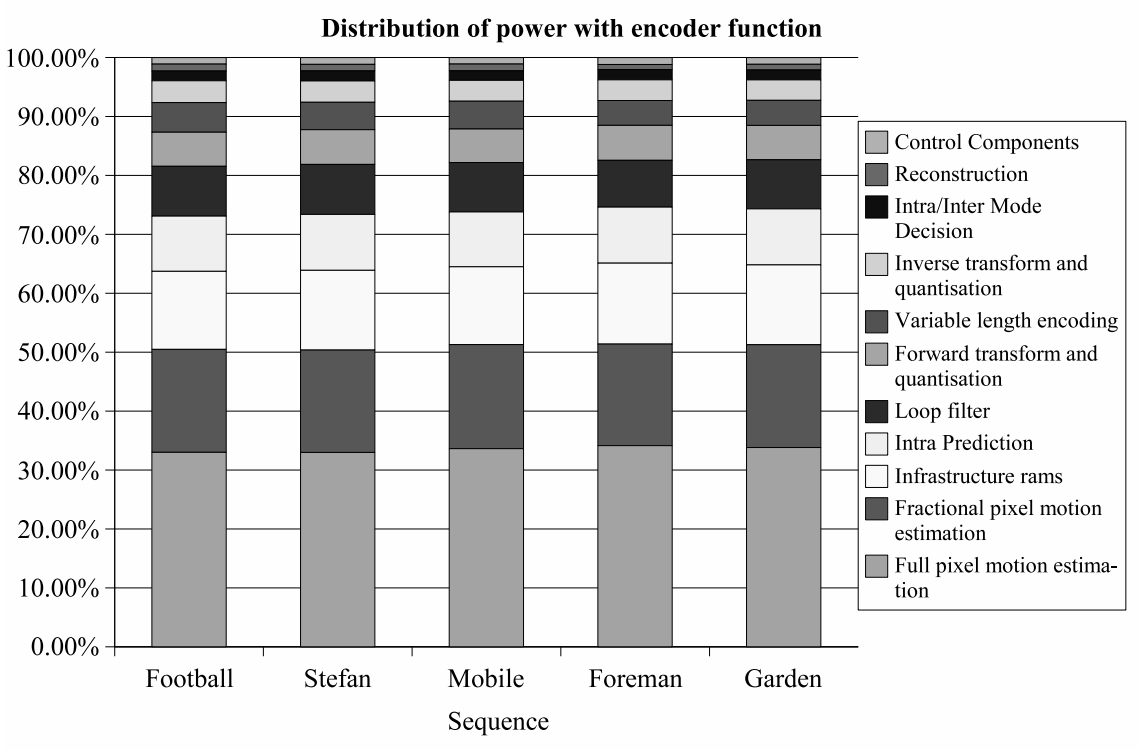

Figure 5.8: Distribution of FPGA dynamic power per encoder function

Encoder Function	Description
Full pixel motion estimation	This preforms the full pixel motion estimation function required by a H.264 video encoder. In this encoder a log search algorithm is used. Two reference frames are used, thus to implement this function two search memories blocks and two motion estimator blocks are used. Only a block-size of 16 by 16 is supported.
Fractional pixel motion estimation	This performs the interpolation and fractional search functions required by a H.264 video encoder. The full fractional search algorithm is used. The fractional estimation process is performed on one reference frame.
Infrastructure rams	Embedded rams required to support encoder operation which are not directly attributable to a single encoder function.
Intra Prediction	This performs the intra prediction function required required by an H.264 video encoder. Only 4x4 intra prediction modes 0-3 are supported.
Loop filter	This performs the loop filtering operation as defined in the H.264 standard.
Forward transform and quantisation	This performs the forward transform and quantisation functions required by an H.264 video encoder.
Variable length encoding	This performs the CAVLC algorithm required by a H.264 video encoder.
Inverse transform and quantisation	This function performs the inverse transform and quantisation operation as defined in the H.264 standard.
Intra/Inter Mode decision	This function determines whether a macroblock is encoded using an intra prediction mode or an inter prediction mode.
Reconstruction	This function performs the addition operation required to obtain reconstructed pixels from the result of the inverse transform and quantisation operation.
Control Components	These functions schedule the loading of data from external memory and control the operation of the other parts of the encoder.

Table 5.3: Description of the various encoder functions

From Figure 5.8, it can be seen that the two motion estimation functions (full pixel and fractional pixel) consume the most power. When combined these two functions account for approximately 50% of the total dynamic power consumed in the FPGA. The full pixel estimator alone accounting for 33% of the total dynamic power. The full pixel motion estimator in the encoder studied uses a log search algorithm, 2 reference frames, and uses only one block size, 16x16. This is a low complexity motion estimation configuration. It requires a minimum of 38400 subtraction, addition, absolute and comparison operations per macroblock, compared to over 1.6 million if the basic full search algorithm was used (assuming a square search range of 32 by 32 pixels). If a more complex, higher performance motion estimation algorithm was used the proportion of power consumed by the full pixel motion estimator would increase even further.

The full fractional search algorithm is used by the fractional pixel motion estimator. The fractional estimation process is only performed using one reference frame. The decision on which reference frame is used is made using the full pixel estimation results. If the reference frame used was based on the fractional search results, as in the JM reference model, the fractional search algorithm would consume more power than the full pixel motion estimator. This shows that, when less complex algorithms are used for full pixel motion estimation, the power used by the fractional search becomes more significant.

The infrastructure RAMs are those which are required to support pipeline operation but are not directly attributable to a single encoder function. Combined they are the third most significant source of power consumption in the encoder. The main component of the infrastructure RAM power consumption is the input RAM. This consumes approximately 40% of the total infrastructure RAM power consumption. The input RAM is used by the two motion estimator functions

and the intra prediction function. This further emphasises the importance of the motion estimation algorithm and architecture, with respect to overall dynamic power consumption. Further discussion of embedded RAM power consumption is given in the next section.

The encoder uses a fast mode decision algorithm to determine if intra prediction is required. However, in the version of the encoder analysed this is not taken advantage of to reduce power consumption. Thus, intra prediction requires approximately 10% of the dynamic power consumed by the encoder. A large proportion of this power consumption could be saved if the intra prediction logic was disabled when it was known that its results would not be used. The saving would be more significant if all the 4x4 intra prediction modes were supported as opposed to only modes 0 to 4, which are supported currently.

The other encoder functions in total consume approximately 25% of the power consumed by the encoder. This suggest that reducing the power in any single one of these functions will not, by itself, have a significant impact on overall encoder power consumption.

Embedded RAMs

The overall results shown in table 5.2 apply when each embedded RAM is enabled only when data is being read from it or written to it. If all the embedded RAMs used are permanently enabled, as was the case initially, they consume a much greater amount of power. Figure 5.9 compares the dynamic power consumed by the embedded RAMs when they are enabled as necessary, and when they are enabled permanently.

Figure 5.9 show how significant embedded RAM power consumption can be in a pipelined encoder. If permanently enabled, 60% of the dynamic power con-

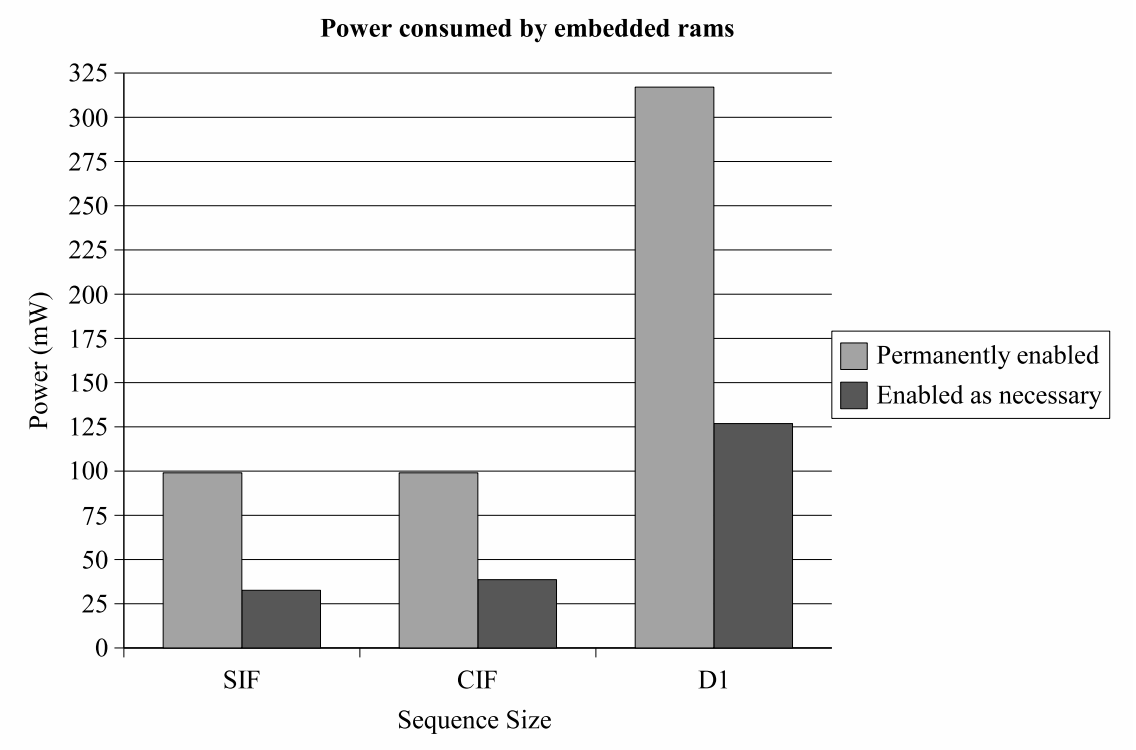

Figure 5.9: Embedded ram power consumption

sumed by the FPGA encoder is consumed within the embedded RAM blocks. Even when enabled as necessary, 35% of the total dynamic power is consumed within them. This is not surprising given the large number of embedded RAMs typically used in a pipelined encoder design. Generating the appropriate signals to enable/disable each embedded RAM incurs minimal resource costs. Given this, appropriate embedded RAM enable signals should be present even when power is not a key design concern. When power is a key design concern the RAM power reduction techniques described in [54] could be used to reduce the proportion of power consumed by the embedded RAMs even further. In general, these were not applied here. Although the effect of applying these techniques to the search memory is examined in the next section.

Motion Estimation

Due to the significance of the full motion estimation function to overall FPGA dynamic power consumption, this component was studied further. While requiring a minimal number of operations, the log search algorithm cannot, in common with many other fast search algorithms, re-use the search area data as efficiently as the full search algorithm. Thus, the search area memory buffer consumes approximately 70% of the power consumed by the full pixel motion estimator as shown in figure 5.10.The results used to generate figure 5.10 are shown in table 5.4.

This is not an unexpected result, a number of studies have noted the significance of the power consumed by the search area memory with respect to ASICs. Fast search algorithms, which attempt to increase search area data reuse, and hence reduce search RAM power consumption, have also been proposed [52]. However, these algorithms have generally been focused on ASIC design. The

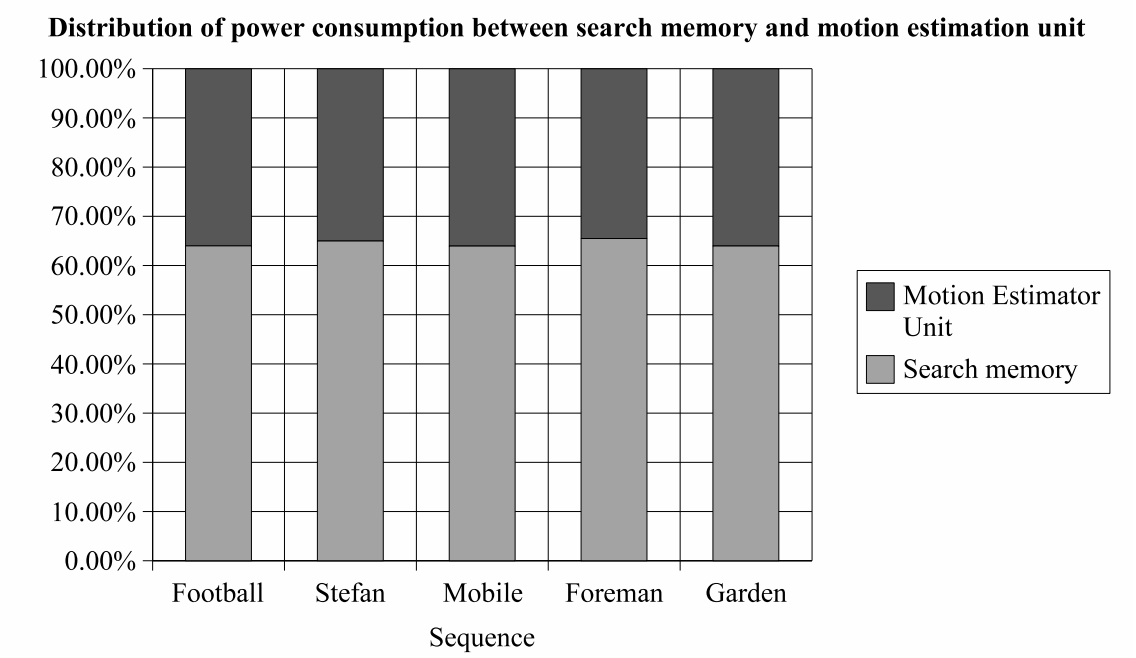

Figure 5.10: Distribution of power between search memory and motion estimator unit

Sequence	Search Memory(mW)	Motion estimator Unit(mW)
Football	6.33	3.56
Stefan	6.18	3.33
Mobile	7.72	4.35
Foreman	7.48	3.94
Garden	24.32	13.7

Table 5.4: Power consumed by search memory and motion estimator block

results presented thus far suggest they could also be of use in FPGAs.

However, these results were determined with all the embedded RAMs (32 M4Ks) used for the search memory being read on each clock cycle. This does not have to be the case. As shown in [54], only the number of embedded RAMs needed to provide the output bit width required must be read in each clock cycle. The motion estimator unit used requires a 32 bit data input. Therefore the memory must have a 32 bit output bit width. This requires a minimum of two M4Ks to be enabled per clock cycle. This would however require a 16-to-1 output multiplexer to select the appropriate M4K outputs. Other combinations, where more embedded RAM are enabled, but a smaller output multiplexer is required could also be used.

As an experiment, a new log search estimator was implemented. This supported a search range of 16x16 thus requiring a smaller number of embedded rams (8 M4Ks). The search memory was implemented using a 4 to 1 multiplexer. Thus, only two embedded RAM blocks required to be active at any one time to provide the motion estimator unit with the 32-bit input it requires.

While the motion estimator unit used in this experiment was a different design to that used initially, there were negligible differences in the power each of the motion estimators consumed. The power used by the search memory however was reduced substantially as shown in table 5.5. This substantially modifies the distribution of power between the search memory and motion estimator unit as shown in Figure 5.11

From the results it would appear that the power used by the search memory can be reduced significantly at relatively little cost. Only when there is a large search area, requiring a large numbers of embedded memories to implement it, will the size of the multiplexer required be a substantive issue. For a low power

Sequence	Search Memory(mW)	Modified Search Memory(mW)
Football	6.33	0.95
Stefan	6.18	0.92
Mobile	7.72	1.16
Foreman	7.48	1.09
Garden	24.32	3.76

Table 5.5: Power consumed by search memory and modified search memory

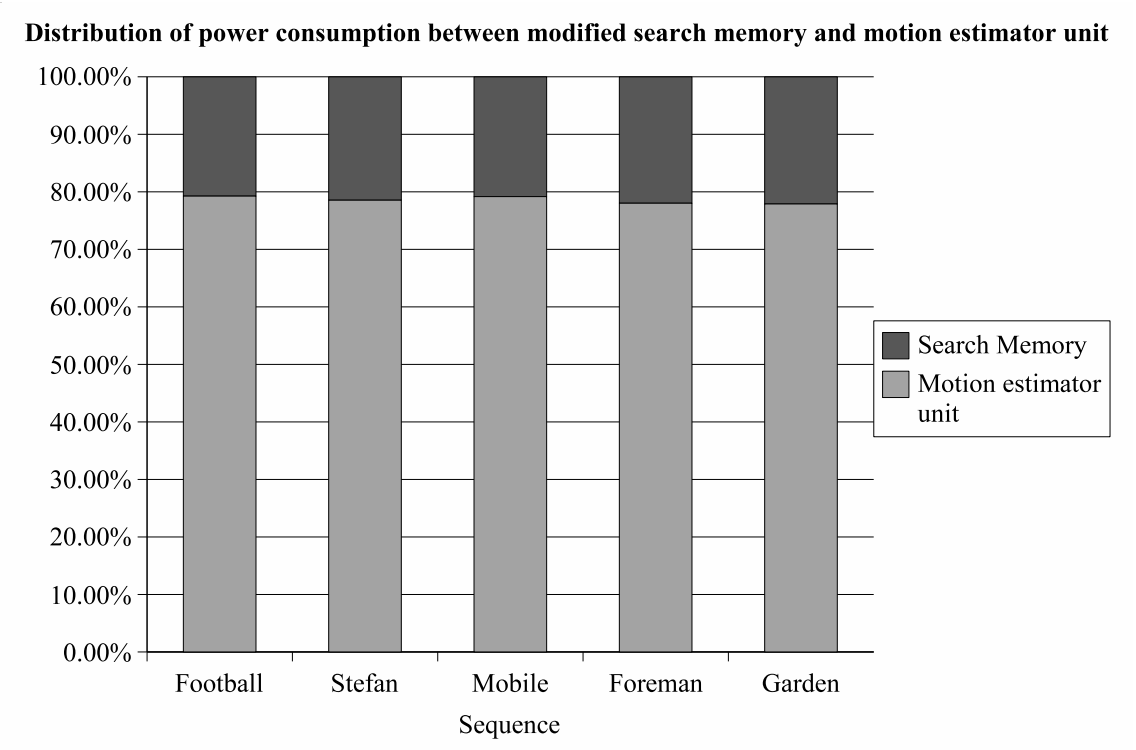

Figure 5.11: Distribution of power between modified search memory and motion estimator unit

FPGA motion estimator therefore, the search area data reuse a motion estimation algorithm/architecture provides should be considered of secondary importance to the number of estimation operations the algorithm/architecture requires.

5.2.4 IO/SDRAM Power Consumption

The encoder uses the level-C data reuse scheme. Therefore, each reference macroblock must be loaded up-to 5 times to support the vertical search range of 32 pixels used. As two reference frames are used, ten reference macroblocks must be loaded per encoded macroblock. As a result, 65% of the IO power is consumed loading search memory data into the FPGA as shown in figure 5.12. A similar percentage of the power used by the SDRAM internally is also consumed loading the search memory data into the FPGA. Complete power results for the IO and SDRAM are given in Appendix D

Given this, the level-D data reuse scheme discussed in section 2.3.2 would appear to offer a substantial power reduction. In this system it would reduce the power used loading search memory data into the FPGA by almost 80%, and reduce the total IO/SDRAM power used by approximately 50%. Offsetting this saving would be an increase in power used by the embedded RAMs on the FPGA. It is also worth considering that the large number of embedded RAMs required to implement the level-D data reuse scheme may force a larger FPGA to be used to implement the encoder. This will increase the static power consumption, and further mitigate the power savings form using the level-D data reuse scheme.

If a smaller search range of 16 pixels and only one reference frame was used, loading the search memory data would consume approximately 35% of the total SDRAM/IO power consumption. The other load and store operations becoming more significant. Of these, the output frame save and load operations could

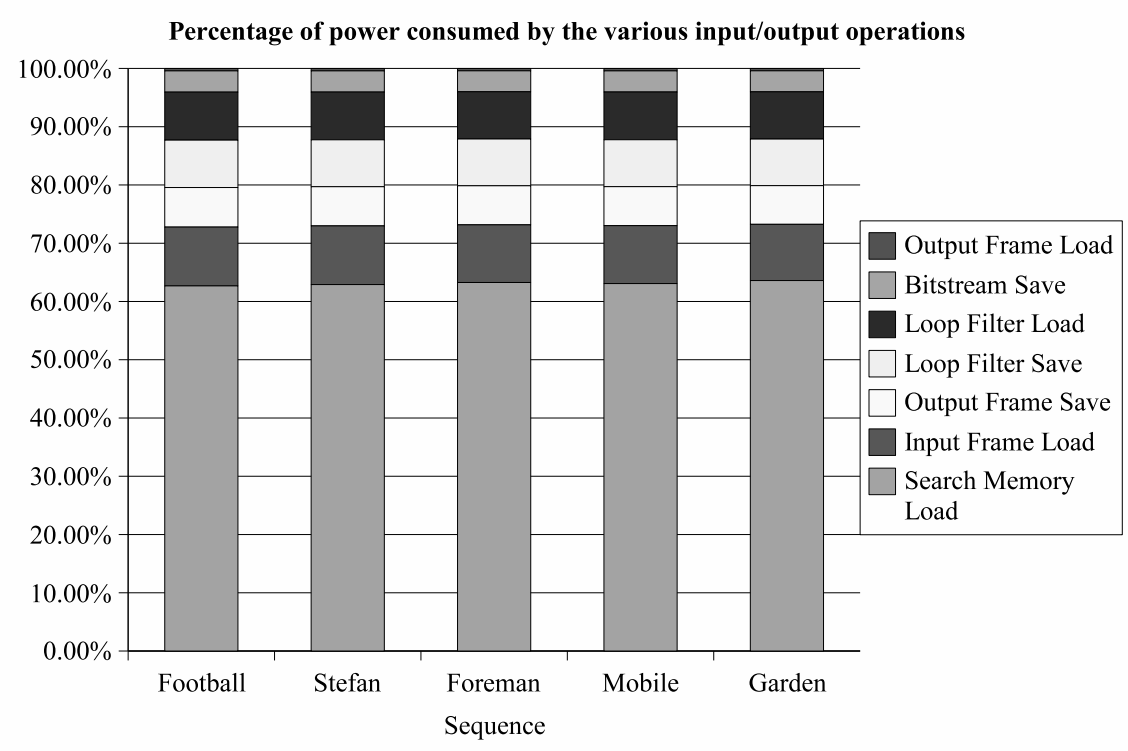

Figure 5.12: Distribution of IO power consumption in encoder

be relatively easily eliminated. In the encoder analysed the loop filter operates independently, loading and saving all frame data it requires directly from external memory. If the loop filter accepted data directly from the end of the encoder pipeline, there would be no need for the output frame load and save operations. The size of the loop filter load operation could also be reduced. The loop filter would still require some data to be loaded however as a result of it needing to filter across macroblock boundaries. In addition a line of output pixels would need to be stored internally within the FPGA to provide the data required for intra prediction.

5.3 Summary

The power consumption of an H.264 encoder implemented on an FPGA has been analysed. It has been shown that static power is significant in an FPGA and cannot be ignored especially when encoding low resolution sequences. With regard to FPGA dynamic power consumption it has been shown that the motion estimation function consumes the most power. The full pixel motion estimation function consuming 33% of the total dynamic power consumption, in the encoder studied. The fractional pixel motion estimation function consuming approximately 17% of the total dynamic power consumption, in the encoder studied.

It was also shown that, the dynamic power consumed by embedded RAMs can be significant in an FPGA based video encoder but, by using simple, low cost, techniques, this source of power consumption can be reduced substantially. Embedded RAM power consumption was further studied in the context of the full pixel motion estimation function. It was shown that, as a result of reducing the power consumed by the search memory, it is the motion estimator itself which

consumes the majority of power in an FPGA implementation. This is a distinct difference compared to ASIC implementations, where the power consumed by the search memory can be significant.

As well as consuming the greatest amount of power internally within the FPGA, a substantial part of the power used communicating with external memory is also attributable to the motion estimation function. This is a result of the large amount of reference frame data needed for the motion estimation function. Overall, the results clearly justify focusing on the motion estimation function, when attempting to reduce the dynamic power consumed by a FPGA based video encoder.

Chapter 6

Using adaptive propagation to reduce the power used by an FPGA video encoder's memory bus

In this chapter an algorithm is proposed to reduce the power consumed by an FPGA video encoder's external memory bus. As shown in chapter 5 this represents a substantial proportion of the power consumed by an FPGA based H.264 video encoder. The algorithm reuses the results of the intra prediction process required in an H264 video encoder. As such the algorithm is specific to video encoding. However, it can also be used in conjunction with other more generic bus encoding algorithms.

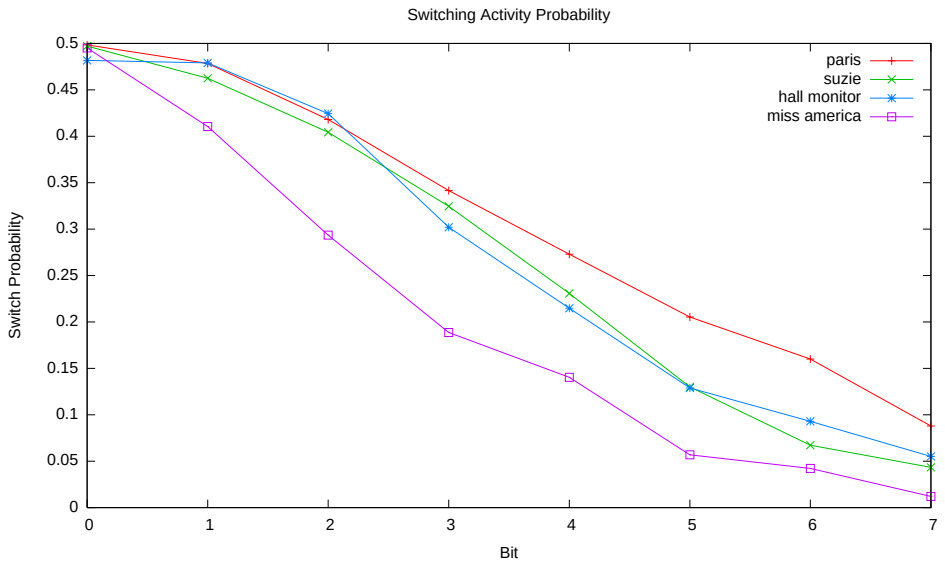

Figure 6.1: Estimated switching probability for various unencoded sequences. The probability of the MSB (bit 7) switching is much less than the probability of the LSB (bit 0) switching

6.1 Algorithm

It is well known that the switching activity associated with video, and other multimedia data, varies dramatically between the least and most significant bits [124]. For typical unencoded video sequences the probability of the least significant bit switching is approximately 0.5. The probability of the most significant bit switching is closer to zero, its actual value being sequence dependent. For the other bits there is a gradual decrease in switching activity as the significance of the bit in question increases as shown in Figure 6.1.

The switching activity probabilities shown in Figure 6.1 were estimated by reading the first 100 frames of each sequence on a row by row basis. If the frames are read on a column by column basis different estimates are obtained, as shown in Figure 6.2. This indicates that altering the direction used to read or process the image data can have an effect on switching activity and hence power consumption.

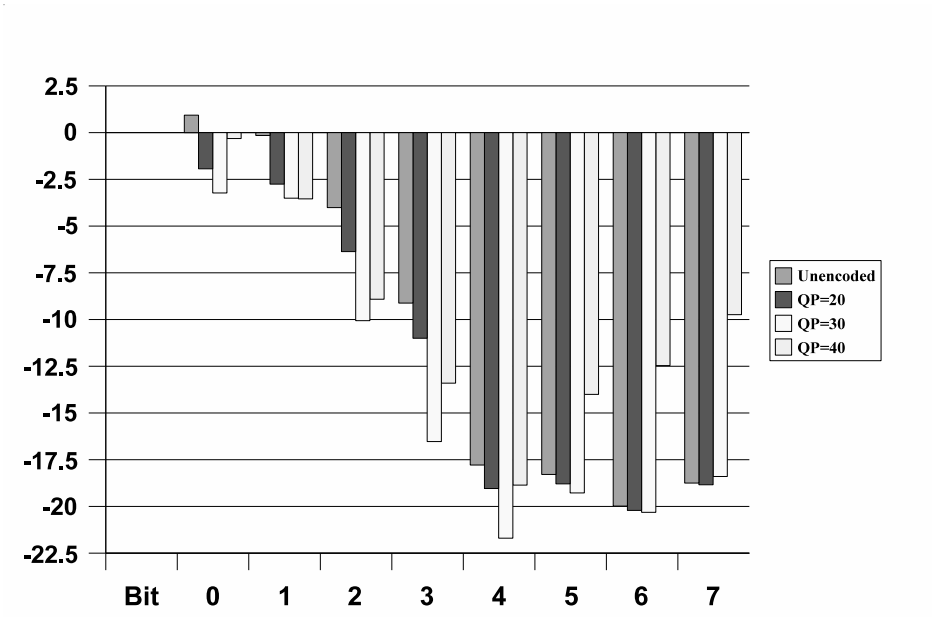

Figure 6.2: Percentage increase in switching activity for the unencoded and various encoded versions of the *suzie* sequence when data is read in the vertical instead of the horizontal direction

The significance of any potential power saving increases when encoded images are considered. The transform and quantisation operations used in H.264 reduce the switching activity on the least significant bits. As shown in Figure 6.2, the least significant bits are less affected by altering the propagation direction. In addition, the transform and quantisation operations tend to remove those transitions least likely to be affected by a change in propagation direction. This accounts for the larger effect altering the direction of propagation has on the least significant bit switching activity of encoded sequences, as shown in Figure 6.2.

The variation in switching activities shown in Figure 6.2 is a direct result of each sequence having differing spatial correlations in the horizontal and vertical direction. In H.264, to improve compression performance for intra coded macroblocks, intra prediction is used to take advantage of any spatial correlation present in the sequence being encoded. The results of the intra prediction stage

can be used to provide an indication of which direction has maximum spatial correlation, and therefore minimum switching activity. Hence the intra prediction results can be used to determine the direction to propagate a macroblock in order to minimise the number of transitions which occur when it is saved to/loaded from external memory. The benefit of using the intra prediction results to make the propagation direction decision is that the decision can be made with minimal additional cost, in terms of both power and area.

In total there are 13 intra prediction modes defined in the H.264 standard; 9 4x4 modes and 4 16x16 modes [20]. In development of the algorithm only the results of the horizontal and vertical prediction modes have been considered as it would be difficult, and costly, to design hardware to support less regular propagation directions. The propagation direction decision could be made at either the frame level, where each macroblock in a frame is propagated in the same direction, or at the macroblock level, where each macroblock's propagation direction is determined independently. Making the decision at the frame level offers a cost advantage. However, the greatest reduction in transitions is achieved if the decision is made at the macroblock level. This is a result of the macroblocks in each frame have differing spatial characteristics.

The vertical and horizontal 16x16 and 4x4 intra prediction modes are shown in Figure 6.3. In the two 16x16 modes each complete macroblock is compared against either the 16 pixels above or the 16 pixels to the left. In the two 4x4 modes each 4x4 sub-block is compared against either the 4 pixels above or the 4 pixels to the left. If attempting to directly determine the best propagation direction for a macroblock adjacent vertical and horizontal pixels would be compared. Given this, the results of the 4x4 modes are more appropriate to use because the maximum distance between the pixels being compared is 4 pixels, as opposed to

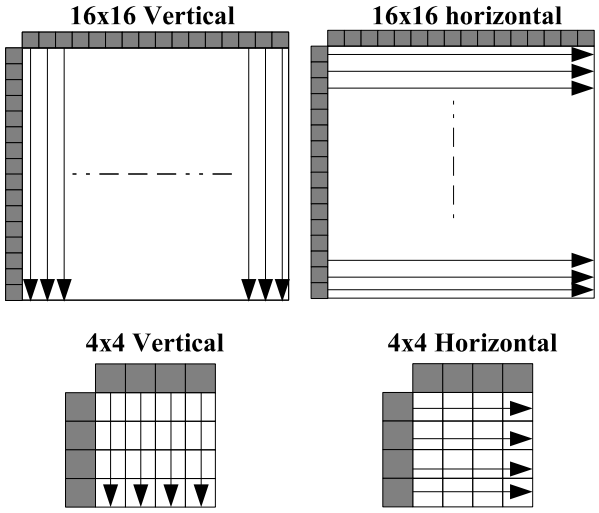

Figure 6.3: 16x16 and 4x4 vertical and horizontal luma prediction modes in H.264.

16 for the 16x16 intra prediction modes. In order to determine the direction in which to propagate a complete macroblock however, it is necessary to combine the individual 4x4 block distortion measures. This creates an additional cost which must be considered.

Algorithm 6.1 Algorithm used to determine the propagation direction from the 4x4 intra prediction results

DMv_i is the 4x4 vertical distortion measure for sub-block i
DMh_i is the 4x4 horizontal distortion measure for sub-block i
if $\Sigma_{i=0}^{i=15} DMv_i < \Sigma_{i=0}^{i=15} DMh_i$ **then**
  propagation direction=vertical
else
  propagation direction=horizontal
end if

The algorithm used to determine the propagation direction for a macroblock when the 4x4 intra prediction results are used is shown in algorithm 6.1. If the 16x16 intra prediction results are used no extra calculations are required. However a comparison, between the 16x16 vertical and 16x16 horizontal intra prediction result, is still needed. Note that in this chapter the Sum of Absolute

Differences is used as the distortion measure. A 32-bit memory bus has been assumed, resulting in the propagation orders shown in Figure 6.4.

There are a number of other generic bus encoding algorithms which have been proposed. Two such algorithms are bus invert [125] and value based mapping(VBM)/difference based mapping(DBM) [126]. The proposed propagation direction reordering algorithm can be used in combination with these other algorithms to provide a larger reduction in transitions.

For the DBM/VBM algorithm, this effect can easily be explained. This algorithm is designed to operated on correlated input data. However, to limit encoding complexity, and the power used performing the encoding operation, it only considers the lag-1 correlation. By using the intra prediction results to adaptively change the propagation direction, the lag-1 correlation on the data inputted to the DBM/VBM encoder is increased. Hence, a greater reduction in transitions occurs as a result.

The bus invert algorithm is sub optimal for the correlated video data being considered. Using a partial bus invert algorithm provides a greater reduction in transitions. Finding the optimum selection of bits to be inverted is an intractable problem [16]. Most benefit is derived when the bus invert algorithm is applied to bus lines whose switching activity is large and/or uncorrelated. In this chapter the bus invert algorithm is applied to the 4 least significant bits of the 8 bit pixel data. By applying the adaptive propagation algorithm in addition to the partial bus invert algorithm, the total number of transitions on the MSB bus lines are also reduced. The result is a larger overall transition reduction.

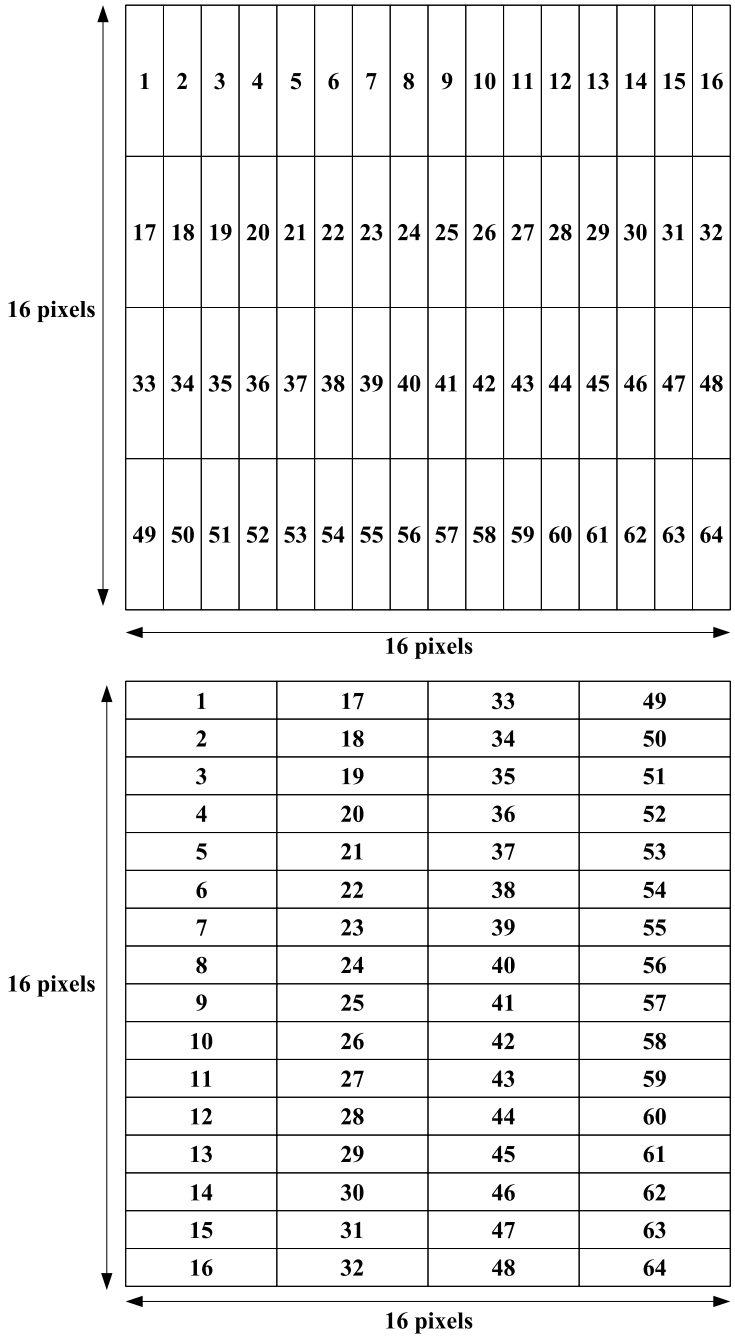

Figure 6.4: Horizontal (top) and vertical (bottom) propagation orders for a macroblock. Each word is 32 bits (4 pixels) wide

6.2 Bus Encoder/Decoder Implementation

There is a practical issue using bus encoding algorithms such as DBM/VBM and bus invert with commodity memory devices that do not have built in bus encoding/decoding hardware [127]. Nearly all bus encoding algorithms rely on knowledge of the previous bus value to determine the current bus value. The codeword value is dependent on the context in which it was encoded. It is difficult to use such algorithms with commodity memory devices because, to obtain the expected transition reduction, the data must be read in the same context it was saved. For the DBM/VBM algorithm, the data cannot even be correctly decoded if it is read in a different context to which it was written.

The issue of context is a problem for the video compression specific bus encoding proposal given in [128]. This uses DBM/VBM, but to exploit the correlation between multiple reference frames, encodes/decodes one reference frame in the context of another. If used with a commodity memory device, encoding reference frames in this way makes it impossible to access them on an individual basis. Thus it would be impossible to use the proposal given in [128] with commodity memory devices. In general, the context issue is mitigated in video compression systems because the video data is typically saved and loaded in macroblock sized chunks. Each word written to memory is loaded in the same context in which it was saved.

Nearly all bus encoding studies have been focused on ASICs. Implementing bus encoding techniques on FPGAs is challenging due to the high power cost associated with FPGA logic. The power used to implement the bus encoding and decoding operations can frequently outweigh the power saved by reducing the transitions on the bus. Only the authors of [129] have studied bus encoding

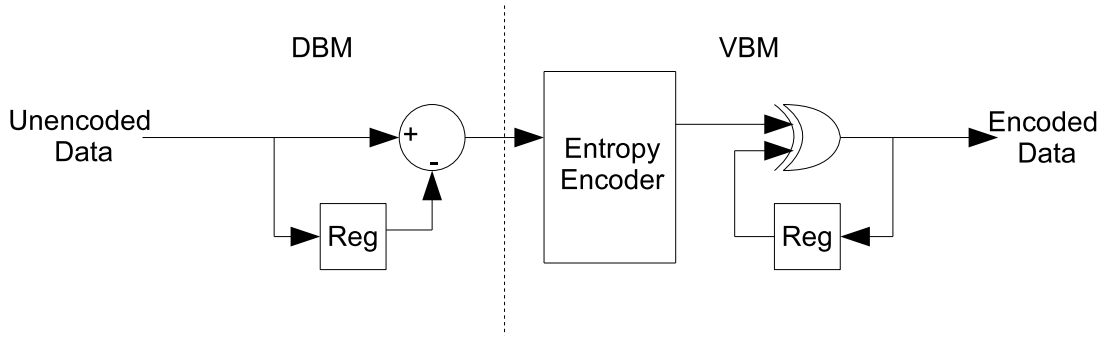

Figure 6.5: DBM/VBM Encoding Process

specifically on FPGAs. The results given in [129] indicate that for their chosen FPGA (A Virtex Device) their algorithm and architecture would not result in an overall power saving for realistic bus capacitance values. In this section, the implementation of DBM/VBM encoder architecture is pursued because as shown in section 6.3.1, this algorithm in combination with the proposed adaptive propagation algorithm provides the greatest reduction in transitions.

6.2.1 DBM/VBM Algorithm Overview

The DBM/VBM algorithm is specifically designed to reduce transitions on correlated input data [126]. A diagram of the DBM/VBM encoding process is shown in Figure 6.5. Difference Based Mapping is the first stage of the encoding process. The purpose of this stage is to remove the redundancy, present in the input data. The DBM algorithm does this by computing the difference between the previous input value and the current input value. The difference is then mapped to a positive value within the range 0 to $2^N - 1$, where N is the bit width of the input data.

The second stage, Value Based Mapping, can be viewed as an entropy encoding process. The aim being to reduce the number of transitions on the VBM

Input	Output
000	000
001	001
010	010
011	100
100	011
101	101
110	110
111	111

Table 6.1: VBM codetable for a bit width of 3

output instead of reducing the output bitrate, as would be the aim in conventional entropy encoding. The XOR gate and register convert any *logic 1* produced by the entropy encoder into a transitions on the output bus. To minimize the number of transitions the entropy encoder matches the number of bits in the output word with logical value of 1 to the probability of an input word occurring. The greater the probability of an input word occurring, the fewer the number of *logic 1* bits in the output word. VBM assumes that the probability of an input word occurring decreases as the value of that word increases. Thus, for VBM the number of ones in the output word increases as the value of the input word increases. The VBM code table for a bit width of 3 is shown in table 6.1. The DBM/VBM decoding process is simply the inverse of the encoding process. More details on the DBM/VBM encoding algorithm are given in [126]

6.2.2 DBM/VBM Implementation

As previously stated a memory bus width of 32 bits has been assumed. This does not require a DBM/VBM encoder and decoder which supports 32-bit operation. The hardware required would be too complex. More importantly it would offer little benefit because the video data will only be correlated, and hence the

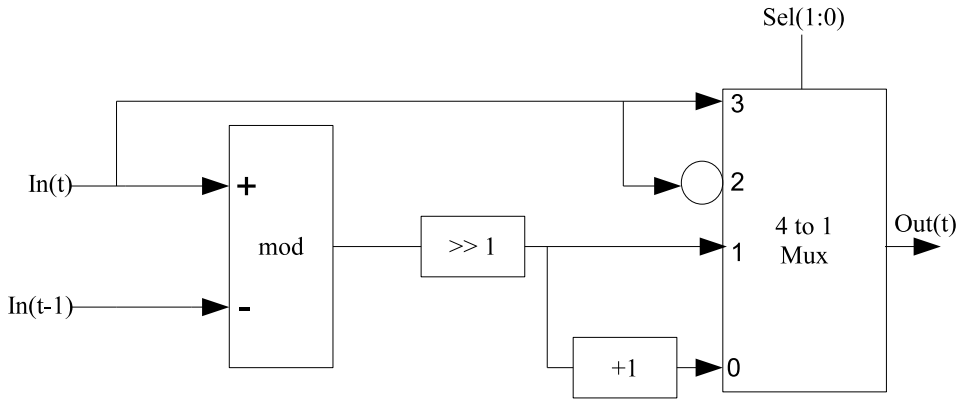

Figure 6.6: Simplified diagram of a DBM encoder

DBM/VBM coding scheme effective in reducing transitions, if it is considered in segments which have a bit width which is less than or equal to the pixel bit width (8 bits).

DBM is primarily an arithmetic operation. As such it can be efficiently implemented using the optimised carry chain logic present in the Cyclone-3 FPGA used. A diagram of the implementation is shown in Figure 6.6. Not shown on the diagram are the control inputs to the 4 to 1 multiplexer. Sel(0) is one if $In(t) \geq In(t-1)$ else it is zero. Sel(1) is one if $|In(t) - In(t-1)| > (In(t-1) \oplus MSB(In(t-1))$ else 0. The DBM decoder is implemented in a similar way.

An algorithm for implementing the VBM operation is given in [126]. Implementing this algorithm would require either multiple clock cycles per result (up to the number of bits in the VBM input/output), or a significant block of combinatorial logic without any registers. Both options are undesirable as they would have a large impact on an overall system, either by reducing the available memory bandwidth, or, by reducing the clock rate the system can operate at. Therefore, only direct combinatorial implementations have been considered.

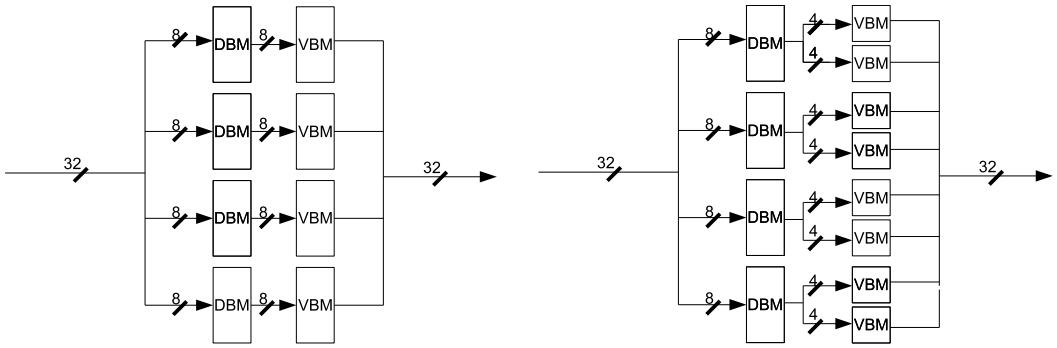

Figure 6.7: Different DBM/VBM Encoding Structures Implemented, 8-bit DBM/8-Bit VBM and 8-bit DBM/4-bit VBM. The decoding structures are similar.

The critical parameter in a direct combinatorial implementation is the bit width of the VBM encoder used. As already mention the maximum useful width is 8 bits. Directly implementing an 8-bit VBM encoder requires significant logic resources, as it requires the eight 8-bit functions to be implemented. A 4 bit VBM encoder can be implemented much more efficiently as each output bit can be produced using only 1 LUT. However, reducing the width of the VBM encoder increases the transitions on the output bus. However, since the overall aim of the encoding process is to reduce the total power used this may not be a great disadvantage given the much higher resource, and hence power cost, of 8-bit VBM encoding. Two DBM/VBM encoding/decoding structures were therefore implemented. One using 8-bit DBM and VBM encoders, the other using 8-bit DBM encoders and 4-bit VBM encoders. A diagram of each structure is shown in Figure 6.7.

6.3 Results and Discussion

6.3.1 Adaptive Propagation Algorithm

To initially test the algorithm's performance it was implemented into the JM 12.2 Reference Encoder [130]. The number of transitions when each macroblock was propagated vertically, horizontally and adaptively based on the 4x4 intra prediction algorithm described was then measured. The results given are the average across the first 100 frames of each sequence

Figure 6.8 shows the percentage reduction in transitions when the adaptive propagation algorithm is used compared to when both fixed horizontal, and fixed vertical, propagation are used. For this experiment, the sequences used were encoded using a constant quantisation parameter (see section 2.1.2) of 30. For all sequences the adaptive propagation algorithm reduces the number of transitions. The extent of the reduction is to a large degree, sequence dependent. The largest reduction in transitions occurs for sequences which contain a mixture of horizontally and vertically correlated macroblocks, such as *hall_monitor*, *paris* and *office*. For these sequences, adaptively changing the propagation direction on a per macroblock basis provides a significant reduction in transitions compared to when either fixed horizontal or fixed vertical propagation is used. For other sequences where the majority of macroblocks are either vertically or horizontally correlated, such as *riverraft* and *suzie*, the adaptive propagation algorithm only provides a small reduction in transitions compared to using the best fixed propagation direction. However, in both cases the adaptive algorithm offers a significant reduction compared to when the worst fixed propagation direction is used. For the rest of the analysis the adaptive algorithm is compared only to fixed horizontal propagation, the best fixed propagation direction for the sequences

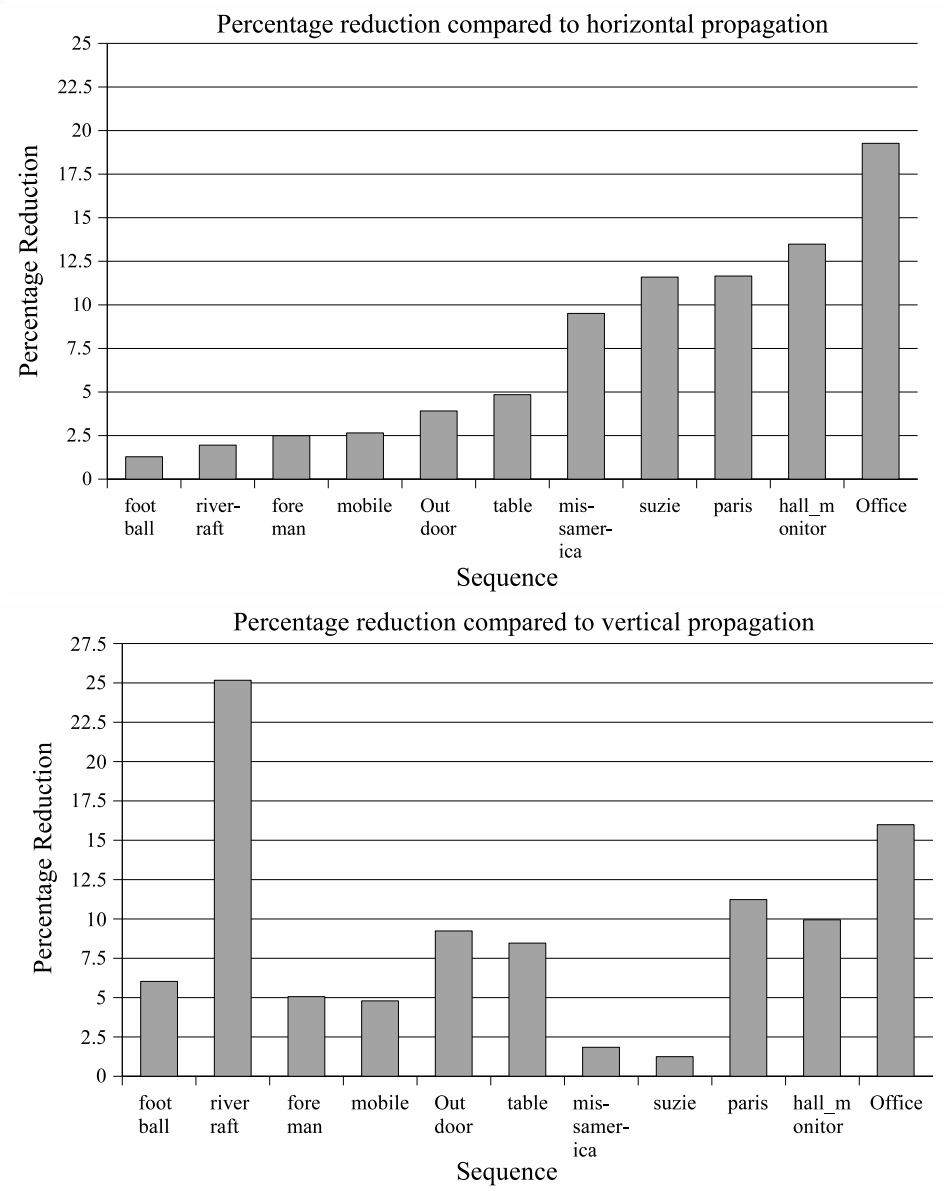

Figure 6.8: Percentage reduction in transitions using adaptive propagation compared to fixed horizontal and fixed vertical propagation

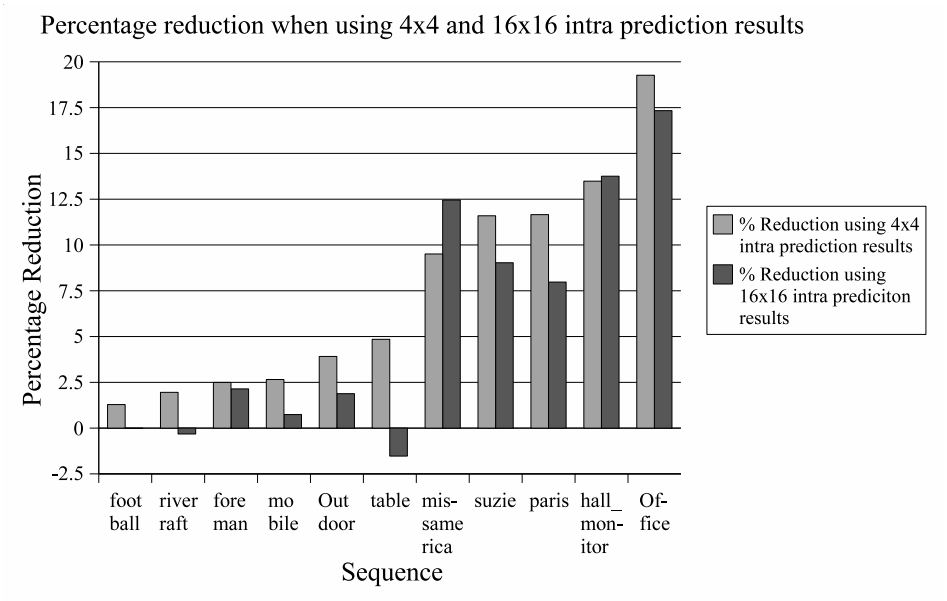

Figure 6.9: Percentage reduction in transitions compared to horizontal propagation when the 4x4 and 16x16 intra prediction results are used

studied.

Figure 6.9 compares the transition reduction achieved using the 4x4 intra prediction results and 16x16 intra prediction results. A quantisation parameter of 30 was again used to encode the images. Apart from two sequences, *missamerica* and *hall_monitor*, using the 4x4 intra prediction results within the adaptive algorithm causes a larger reduction in transitions. More significantly for the *football*, *table* and *riverraft* sequences using the 16x16 intra prediction results actually causes an increase in transitions. The worst case being the *table* sequence. For this sequence using the 16x16 intra prediction results causes a 1.5% increase in transitions compared to when fixed horizontal propagation is used.

Figure 6.10 shows the effect varying the quantisation parameter has on the percentage reduction in transitions achieved by the proposed algorithm. The percentage reduction in transitions does increase as the quantisation parameter increases. At higher quantisation levels, this effect is offset by the inaccuracy

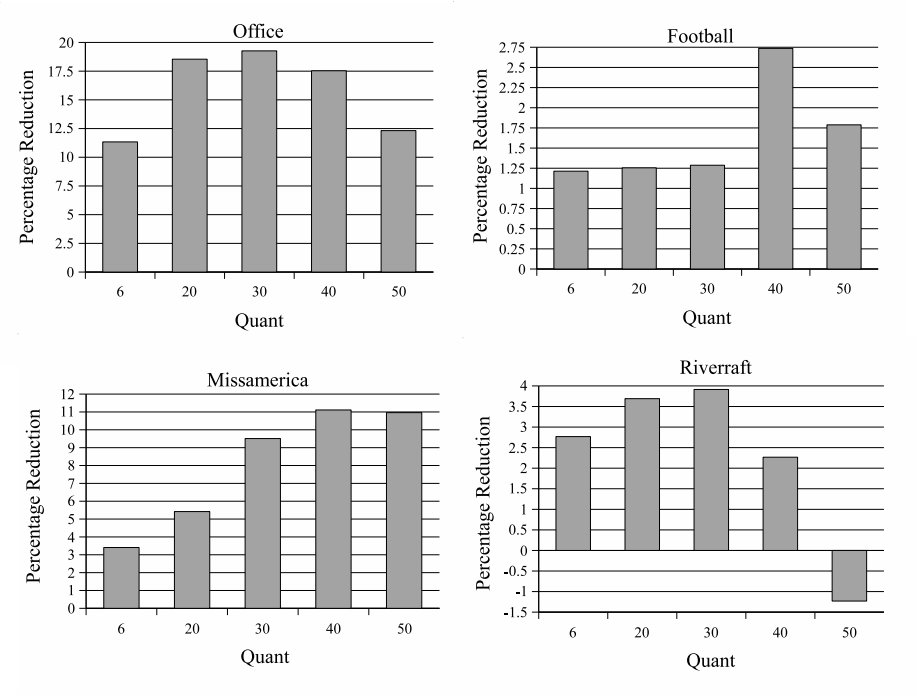

Figure 6.10: Percentage reduction in transitions compared to horizontal propagation for a range quantisation parameters

of the intra prediction results. The intra prediction process is performed on unencoded macroblocks. However, it is encoded macroblocks which are saved to and loaded from external memory. The proposed algorithm implicitly assumes the unencoded and encoded macroblocks will be similar. When a high quantisation parameter is used there is the potential for this assumption not to hold and an incorrect propagation decision to be made. For sequences where the adaptive algorithm does not offer an inherent benefit, such as riverraft, this effect can result in the adaptive algorithm increasing transitions. This only occurs at when the quantisation parameter is set to a very high value, and consequently the encoded images are of very low quality.

The effect of using the proposed algorithm with DBM/VBM encoding and bus invert encoding is shown in figures 6.11 and 6.12. For these experiments a

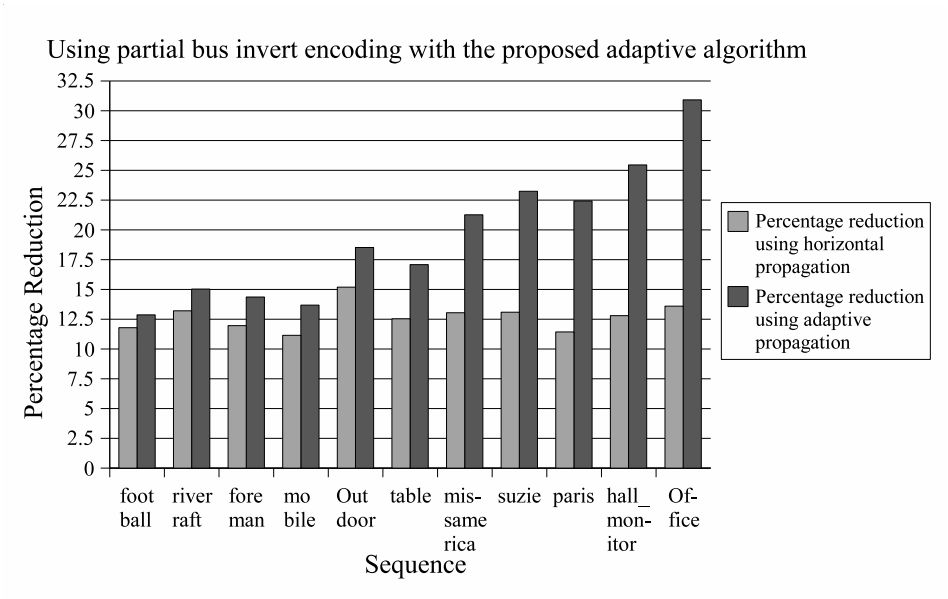

Figure 6.11: Percentage reduction in transitions (compared to fixed horizontal propagation), using partial bus invert encoding with and without the proposed adaptive propagation algorithm

constant quantisation parameter of 30 was used. The results show that using the proposed algorithm in combination with either partial bus invert or DBM/VBM encoding provides a greater reduction in transitions than using bus invert or DBM/VBM encoding alone. The performance improvement is most notable when partial bus invert encoding is used. Using the adaptive propagation algorithm with bus invert encoding more than doubles the transition reduction achieved for sequences where the adaptive propagation algorithm alone offers a significant benefit. Despite this improvement, partial bus invert encoding still offers a smaller reduction in transitions than DBM/VBM encoding.

The performance improvement provided by the adaptive propagation algorithm is less notable when DBM/VBM encoding is used. With DBM/VBM encoding the number of transitions only decreases if the difference between consecutive input values is increased by set amounts, dependent on the size of the

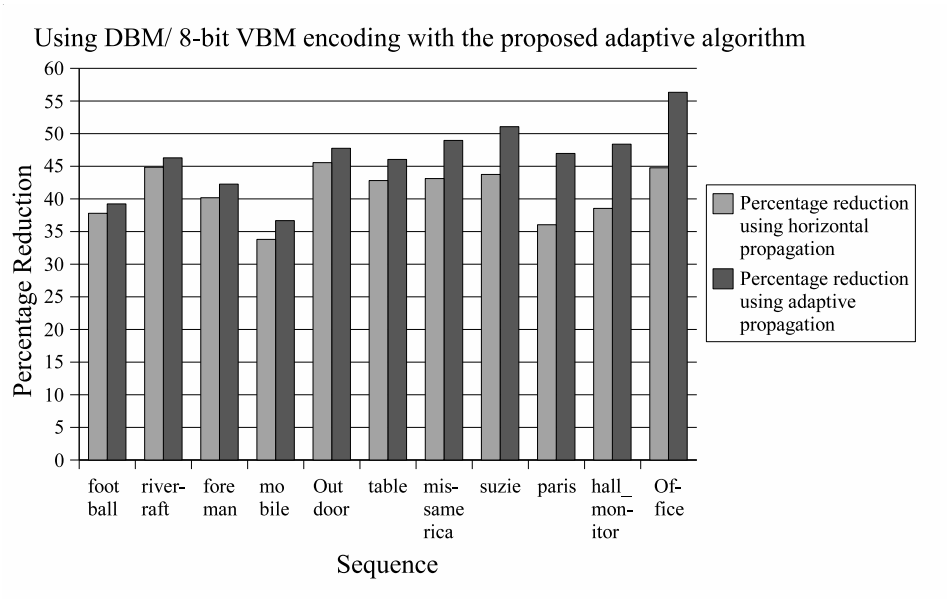

Figure 6.12: Percentage reduction in transitions(compared to fixed horizontal propagation) using DBM and 8-bit VBM encoding with and without the proposed adaptive propagation algorithm

VBM encoder used. Thus when an 8-bit VBM encoder is used, the impact of increasing the lag1-correlation using the proposed adaptive propagation algorithm is limited. However in the best case, using adaptive propagation still provides an additional 10% reduction in transitions. If a 4 bit VBM encoder is used the impact of adaptive propagation is greater as shown in Figure 6.13. The adaptive propagation algorithm providing upto a 15% reduction in transitions.

6.3.2 DBM/VBM Implementation Results

The resources used by both the 8-bit DBM/8-bit VBM and 8-bit DBM/4-bit VBM structures are shown in Table 6.2. From Table 6.2 it can be seen that the 4-bit VBM structure requires approximately half the resources of the 8 bit VBM structure. A near 50% resource saving is obtained for an average 20% increase in transitions.

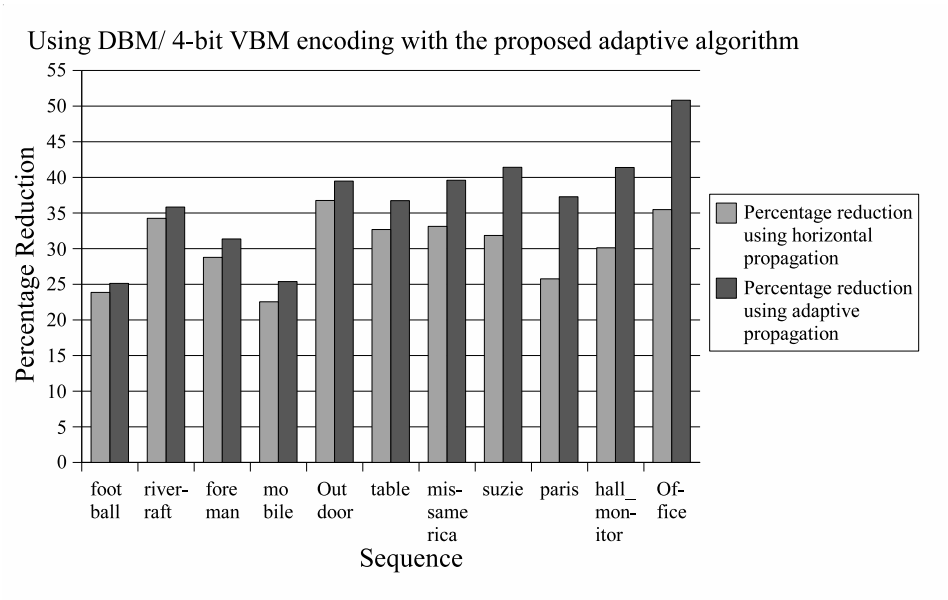

Figure 6.13: Percentage reduction in transitions(compared to fixed horizontal propagation) using DBM and 4-bit VBM encoding with and without the proposed adaptive propagation algorithm

Structure	LUTs	Registers
8-bit DBM/8-bit VBM	696	64
8-bit DBM/4-bit VBM	348	64

Table 6.2: Resources used by the different DBM/VBM Structures Implemented

To estimate the power used performing the bus encoding and decoding operations the Quartus 2 power analyser was used. The power simulations performed to provide input data for the analyser modeled how the encoder/decoder hardware would be used in a video compression system. As such, the bus encoder encoded macroblocks once. The bus decoder decoded macroblocks multiple times to reflect the typical reference frame data reuse strategy employed by a FPGA video encoder (see section 2.3.2). For this study, the video compression system is assumed to have a search range of 16 in both the horizontal and vertical direction, therefore each macroblock was read from memory and decoded 3 times. The clock frequency used during the simulations was 50 megahertz, and the video frame rate was assumed to be 30 frames per second in each case. The results are shown in table 6.3. From table 6.3, it can be seen that there is a substantial power saved when a 4-bit VBM implementation is used compared to an 8 bit VBM implementation. This is a result of the 4 bit VBM implementation requiring both less logic resources and less interconnect resources.

To estimate the power consumed on the memory bus, and hence the total power used, the formula given in equation (3.1) is used. In this study the result given by equation (3.1) is multiplied by four. This is reflect the assumption that the level-C data reuse scheme is used with a vertical search range of +/- 16 pixels. Figure 6.14 shows the total power consumed as a function of bus capacitance on a 2.5V and a 1.8V bus for the *paris* sequence using a quantisation parameter of 30.

From the graphs it can be observed that using DBM and 8-bit VBM encoding uses too much power to be useful in the majority of circumstances. Using DBM and 4-bit VBM encoding in addition to propagation reordering does result in a power saving. However, this is only for a 2.5V bus with a relatively high bus

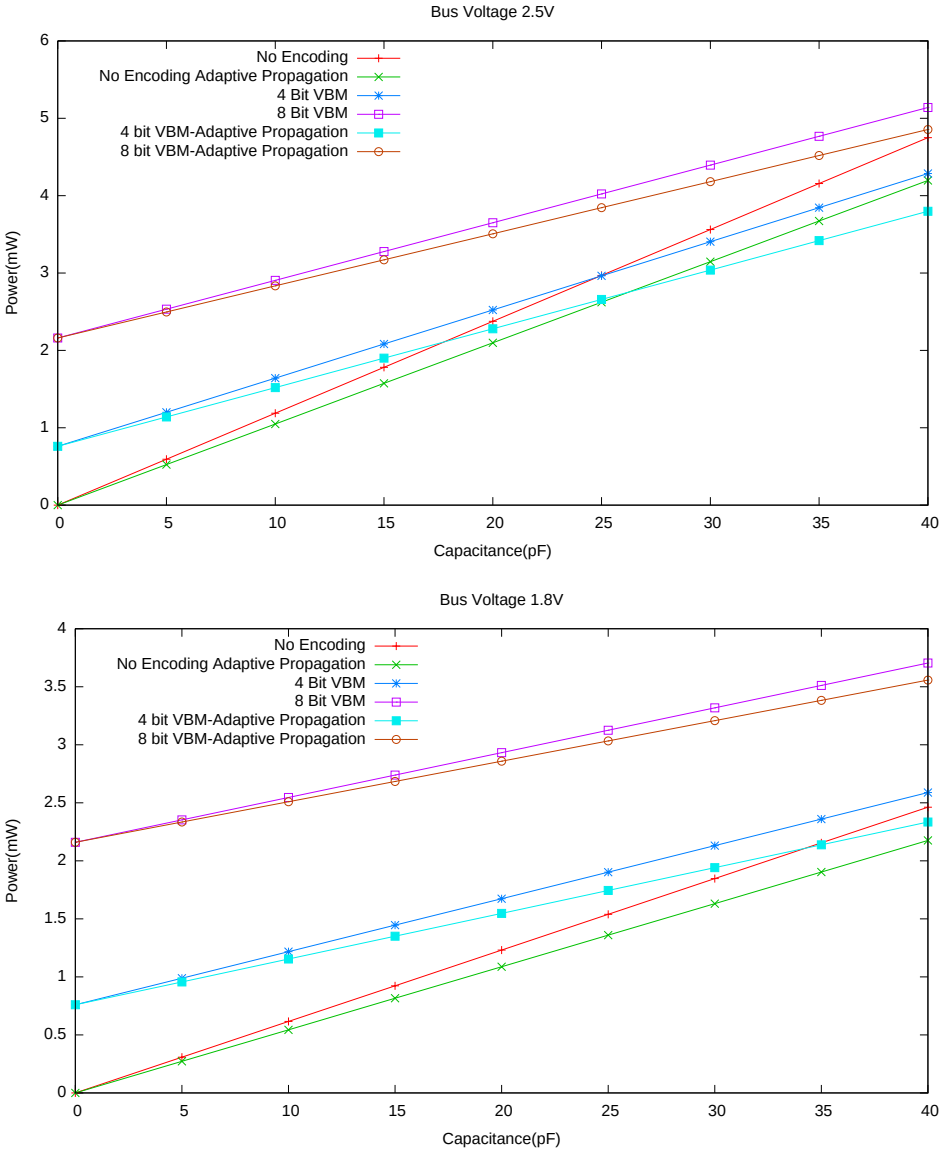

Figure 6.14: Power used as a function of bus capacitance for a 2.5 and 1.8 volt bus

Sequence	Quant	VBM 4-Bit	VBM 8-Bit
Table (QCIF)	6	0.28	0.78
	20	0.28	0.72
	30	0.24	0.63
	40	0.19	0.49
Suzie (QCIF)	6	0.3	0.76
	20	0.28	0.72
	30	0.25	0.65
	40	0.22	0.56
Football (SIF)	6	0.76	2.09
	20	0.71	1.95
	30	0.63	1.74
	40	0.39	1.12
Paris (CIF)	6	0.89	2.49
	20	0.86	2.39
	30	0.76	2.16
	40	0.61	1.76

Table 6.3: Total power used performing the bus encoding and decoding operations (mW)

capacitance. Using solely adaptive propagation results in a power saving in all circumstances. This is under the assumption that it requires negligible power to implement. This is not realistic. Making the propagation direction decision will require some power, as will the reordering operation itself. The exact cost will be dependent on the video compression architecture used. The adaptive propagation algorithm's implementation into a pipelined video compression system is considered in chapter 7.

6.4 Summary

In this chapter an algorithm to reduce the power used by an H.264 encoder's memory bus has been proposed. By reusing the intra prediction results to adaptively the change the propagation order each macroblock is written to and loaded

from memory the number of bus transitions can be reduced by up to 20%. However the performance of the algorithm is heavily dependent on both the sequence and quantisation parameter used.

It has been shown that the proposed algorithm can work in conjunction with both the partial bus invert and DBM/VBM bus encoding algorithms. Combining adaptive propagation and DBM/VBM encoding provides the greatest reduction in transitions. However, results indicate that this scheme is probably of limited practical use in an FPGA. This is due to the relatively large amount of power used in performing the DBM/VBM encoding and decoding operations.

The cost of implementing the proposed algorithm is dependent on the overall architecture of the video compression system, as discussed in chapter 7.

Chapter 7

Integration of the adaptive propagation algorithm into a pipelined video encoder

In this chapter, the implications of implementing the algorithm proposed in chapter 6 within a pipelined encoder architecture are considered. Both a full search variable block size motion estimator and full fractional search motion estimator are designed. Results are given showing that as well as reducing the power used accessing the external memory, the proposed algorithm can also be used to reduce the dynamic power consumed by both the full and fractional pixel motion estimation encoder stages. Power is reduced in these stages by adaptively changing the direction reference data is propagated through them.

7.1 Encoder Architecture

The starting point for the implementation is the encoder described in [117]. It is a typical pipelined encoder consisting of multiple function units separated by RAM blocks. It uses a fixed 1400 clock cycles per pipeline stage. Although the encoder in [117] supports the loop filter operation, to simplify the design its inclusion into the modified encoder has not been considered. To implement the proposed algorithm, it is necessary to have an architecture which can write out reference pixels in either the vertical or horizontal direction, and utilise reference data which has been previously been written to memory in either the horizontal or vertical direction. In addition, the 4x4 intra prediction results need to be available for every macroblock in the sequence.

The cost associated with meeting the intra prediction result availability requirement is dependent on the mode decision algorithm used by the encoder. If the encoder skips the 4x4 intra prediction calculations frequently then meeting this requirement will clearly represent a large additional cost. The encoder studied uses a fast algorithm to determine whether or not 4x4 intra prediction is required. However, the intra prediction results required by the adaptive propagation algorithm are still calculated. The algorithm used is shown in algorithm 7.1. Only unencoded pixels are used in the calculation of $DMuh_i$ and $DMuv_i$ due to the intra prediction availability issue which arises in a pipelined encoder (refer to section 2.4). The use of only unencoded data may adversely effect the adaptive propagation algorithm's performance. However, any impact will be negligible in comparison with the cost of calculating intra prediction results purely for use by the adaptive propagation algorithm. Others have proposed similar intra/inter mode decision algorithms. For example [73] use the diagonal down left, diagonal

Algorithm 7.1 Algorithm used to determine whether 4x4 intra prediction is required

$DMuh_i$ is the 4x4 horizontal intra prediction distortion measure for sub-block i calculated using *only* unencoded pixels
$DMuv_i$ is the 4x4 vertical intra prediction distortion measure for sub-block i calculated using *only* unencoded pixels
DMm is the best motion estimation distortion measure.
if $\Sigma_{i=0}^{i=15} min(DMuh_i, DMuv_i) < DMm$ **then**
  4x4 intra prediction required
else
  4x4 intra prediction not required
end if

down right and DC 4x4 intra prediction results in addition to the horizontal and vertical results within a fast mode decision algorithm proposal.

Data is written into the output memory buffer at a rate of one pixel per clock cycle. Therefore, it is trivial to provide the required output functionality. Only the order in which the buffer write addresses are generated requires to be changed. This can easily be implemented using a multiplexer on the buffer write address.

Utilising the reference pixel data, that has previously been written to external memory, is significantly more complicated. Multiple macroblocks, which may have been written out using different propagation directions, must be read into the search area memory. For the motion estimator to utilise this data effectively, it must be able to read the entire search area using the same propagation direction. It is impractical to have a single pixel data-path from external memory to the search memory or from the search memory to the motion estimator. Requiring the motion estimator to access the reconstructed data on a one pixel per clock cycle basis would increase the macroblock latency associated with the motion estimation operation. In turn, this would increase the frequency at which the encoder has to operate at for a given throughput and reduce hardware efficiency

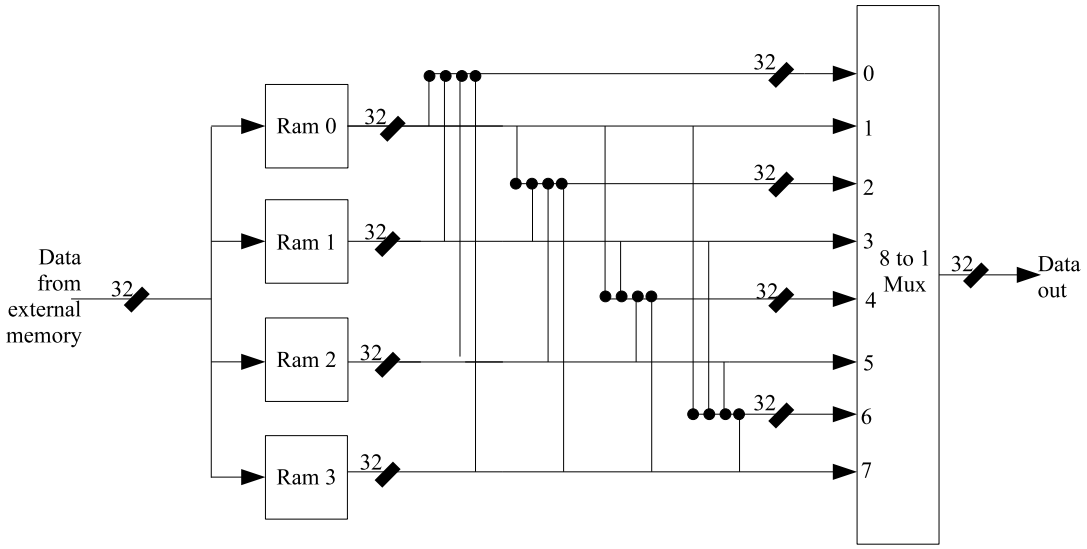

Figure 7.1: Simplified diagram of search memory architecture

(other encoder stages would be forced to wait for the motion estimator to finish). Thus, address modification alone cannot be used to reorder the reference frame data sufficiently for it to be of use to the motion estimator.

7.2 Search Memory Architecture

In FPGAs the search memory is constructed from multiple embedded RAMs. This can be utilised to implement a search memory which can support a data-path width which is greater than a single pixel and also reorder the data as required.

A simplified diagram of the search memory architecture is shown in Figure 7.1. Four separate embedded RAMs are used. Each macroblock is assumed to be loaded into the search memory in one of the orders shown in Figure 6.4. Each four consecutive 32-bit words loaded from external memory are saved to a different embedded RAM. This ensures that each column (horizontal propagation) or row

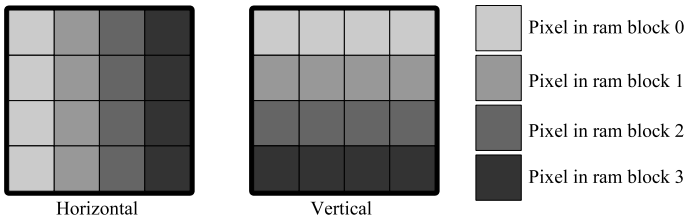

Figure 7.2: Arrangement of each 4x4 block in the search memory when a horizontally and vertically propagated macroblock is loaded into it

(vertical propagation) of each 4 by 4 search area sub-block is stored in a different embedded RAM as shown in Figure 7.2.

To read the desired row or column of the 4x4 sub-block requires either, a read of the appropriate embedded RAM, if the propagation direction the data is written in matches the desired read propagation direction, or a read of all the embedded rams and the selection of the appropriate byte from the 32-bit embedded RAM outputs, if it does not. To minimise the number of clock cycles that each embedded RAM is active, a read operation on all 4 embedded RAMs is only initiated if the motion estimator requires access to a different 4x4 block to the one currently being accessed.

To enable the appropriate pixels to be selected an 8 to 1 multiplexer is required as shown in Figure 7.1. This additional cost is not as significant as it first appears. As shown in section 5.2.3 the most power efficient method of implementing a memory larger than the embedded RAMs available on an FPGA device requires a multiplexer on the output path. The addition of reordering functionality only increase the size of the multiplexer required. The search memory could support a large output bit width using the same multiple embedded RAM technique. However, the resource costs associated with supporting the adaptive propagation algorithm would be greater. For a 64-bit output bit width for example, eight separate embedded RAMs and a 16 to 1 multiplexer would be required.

7.3 Full Pixel Motion Estimation Architecture

Using the proposed search memory architecture it would be possible to implement a motion estimator which does not adaptively change the direction data is propagated through it. However, by adaptively changing the way data is propagated through the motion estimator, additional power can be saved at minimal additional cost. This can potentially offset the power cost of implementing the proposed adaptive propagation algorithm.

A diagram of the full pixel motion estimation architecture is shown in Figure 7.3. The full search algorithm is used due to its high quality and ease of hardware implementation. Current frame data is loaded into the motion estimation array at the start of the motion estimation operation. Reference frame data is propagated through the systolic array to enable the SADs values for each block size and search position to be computed. The decision logic unit applies a bias to the SAD values outputted from the array, and determines which search position has the best cost for all 41 sub-blocks. Finally the best motion vector, and the associated cost value for each sub-block, is written to the motion vector FIFO for use by the fractional pixel motion estimator.

7.3.1 Consequences of supporting adaptive propagation

For the motion estimation operation to be performed successfully current pixel data must be loaded into it in the same direction that reference pixel data is propagated through it. Thus, to support adaptive propagation through the full pixel motion estimator, it must be possible to reorder the current frame data prior to the motion estimation stage of the encoder when required. In the encoder used, this reordering operation can be implemented easily and with minimal

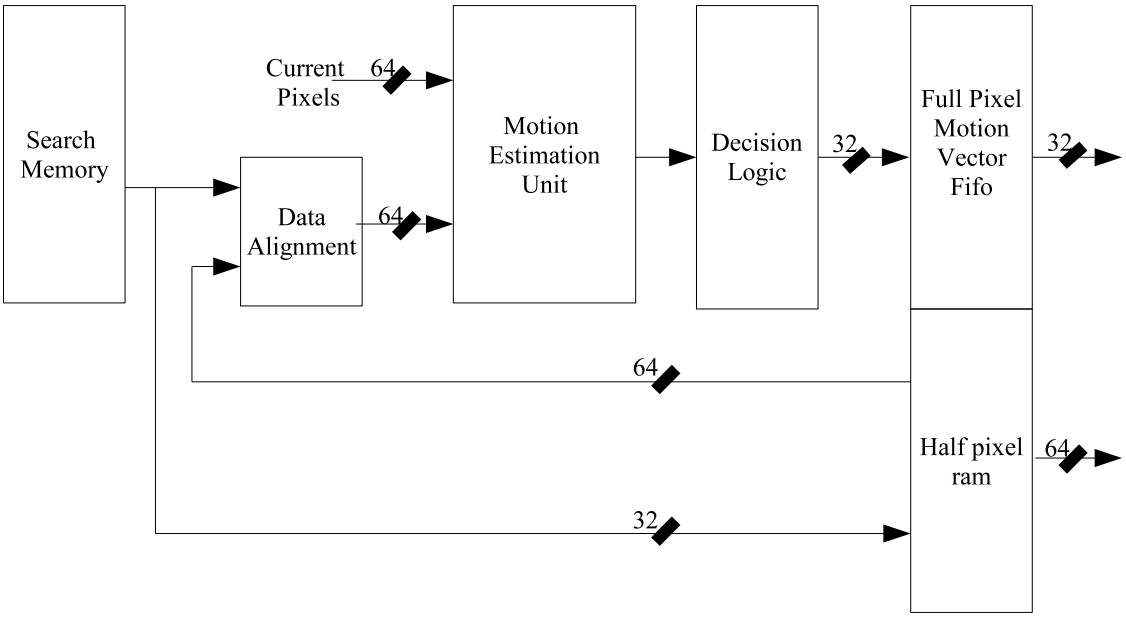

Figure 7.3: Full search motion estimation architecture

additional cost. As shown in Figure 5.1 the first pipeline stage of the encoder is the intra mode decision calculation ($\Sigma_{i=0}^{i=15} min(DMuh_i, DMuv_i)$). While it could be included in a later pipeline stage, this calculation is brought forward because it allows the memory that each input macroblock is loaded into from external memory to be a simple dual-port memory, easily implemented using a single embedded RAM. This is advantageous because it makes it simpler to implement asynchronous external memory and core clocks. Current pixel data is copied from this memory into the main input memory, which provides access to current pixel data for the other pipeline stages which require it (full pixel motion estimation, fractional pixel motion estimation and intra prediction). Data is copied into the main input memory at a rate of one pixel per clock cycle. Thus, reordering of the input data only requires a change in the order the main input memory addresses are generated. As discussed previously this is trivial to implement.

Changing the direction data is read into the motion estimator forces the order

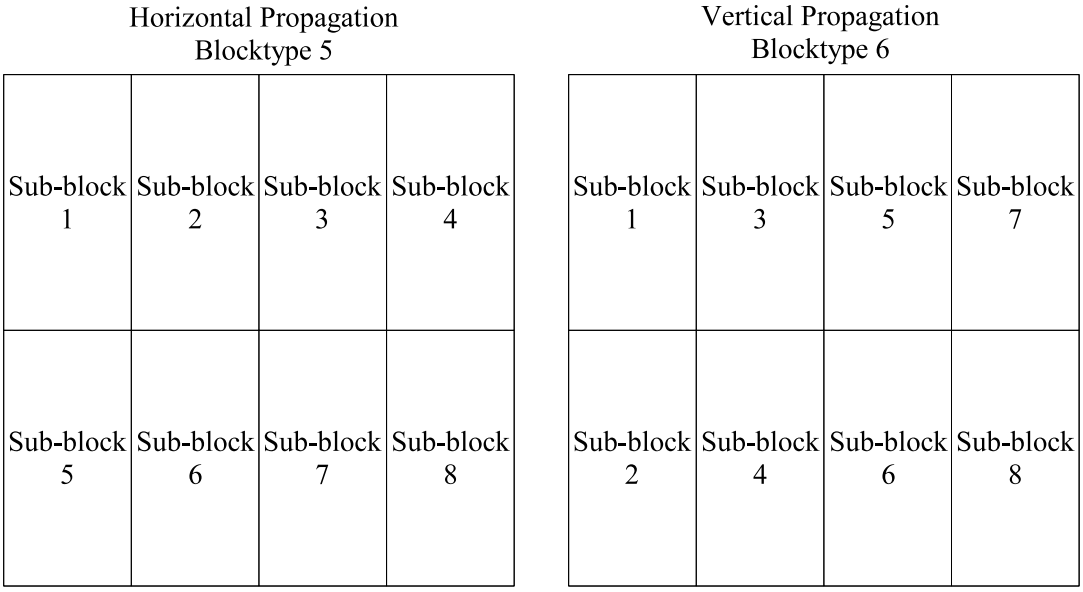

Figure 7.4: Different sub-block indices and blocktypes for a block size of 4x8 when horizontal and vertical propagation are used

in which the search positions are evaluated to be changed also. The search positions being evaluated in horizontal rows if horizontal propagation is used, and vertical columns if vertical propagation is used. This ensures that a similar data flow is used and an identical level of data reuse is maintained regardless of which propagation direction is used. Additionally, when supporting variable block sizes, the correspondence between sub-block indices and sub-block location varies depending on the propagation direction used. For block sizes where the width and height are not equal the correspondence between block size and block type is also modified. These differences are illustrated for the 4x8 block size in Figure 7.4. As the same propagation direction is used in both the full and fractional motion estimation stages, the correct block type and block indices only need to be resolved after the result of the motion estimation process is complete.

7.3.2 Dataflow

In any encoder the search memory data must be available to both the full and fractional pixel motion estimation stages. The previous encoder [117], which only supported a block size of 16x16 pixels, copied the data required for fractional estimation into a separate memory after the full pixel motion estimation operation had been completed. As the full pixel vector was known at this point, only a small fraction of the data present in the search memory needed to be copied. When supporting variable block sizes, a much larger amount of data needs to be copied for use by the fractional pixel motion estimator.

If the same strategy is used that was used previously, a significant proportion of the clock cycles available for the full pixel motion estimation operation would be spent copying data. To prevent this the search data is copied into the half pixel RAM as the full pixel motion estimation process is performed. This requires that the entire search area, and at least a +/- 3 pixel area outside it, be copied into the half pixel memory. For the search range of (-8,+7) used, copying the entire search area reduces the size of the half pixel memory compared to what would be needed if the data required for each sub-block was copied separately. Data from the search memory is aligned on the 4 pixel boundary. To ensure all the required data is copied to the half pixel memory a small amount of extra data is copied, as indicated in Figure 7.5. If horizontal propagation is used the additional data increases the height of the region copied. If vertical propagation is used the additional data increases the width of the region copied.

A secondary benefit copying the data provides is that it allows the motion estimation array to have access to a larger bit width that it would do if it only accessed the reference frame data directly from the search memory. As shown in Figure 7.3 the data copied to the half pixel memory is reused by the full pixel

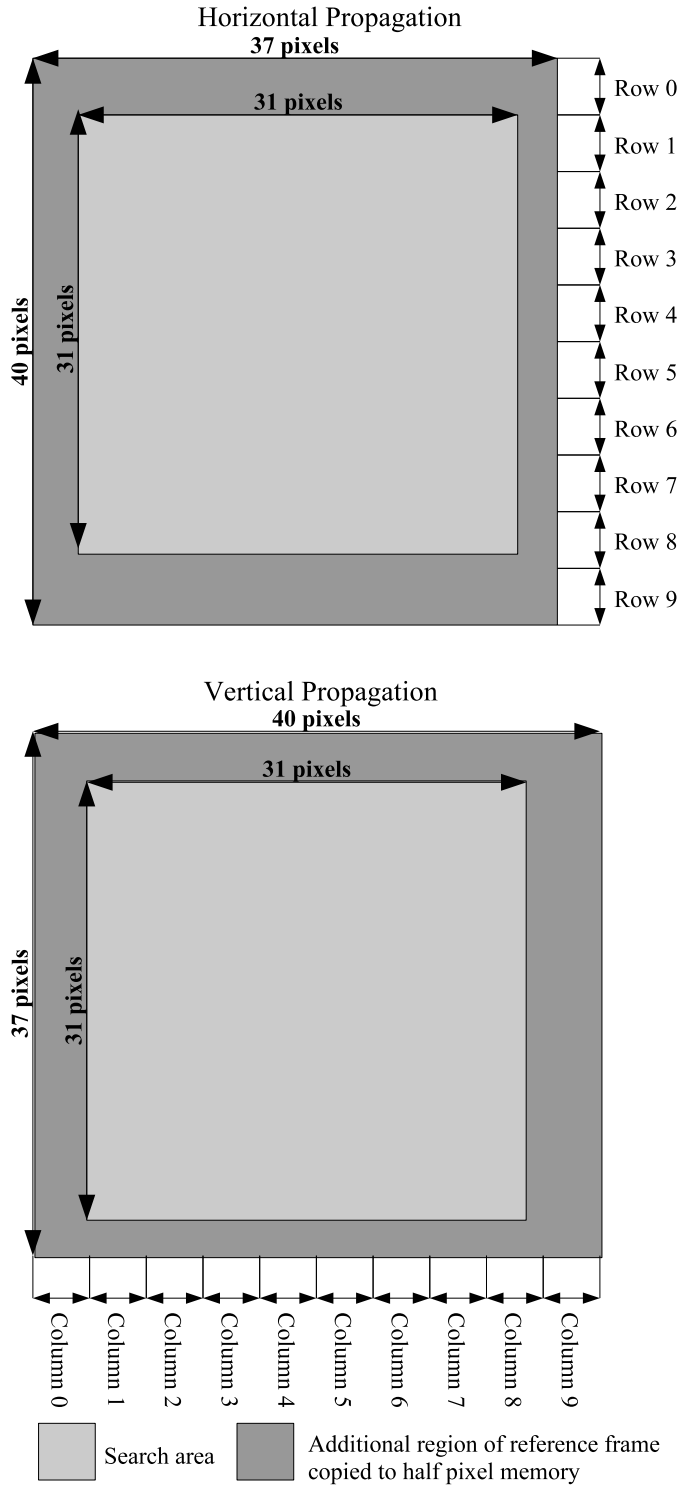

Figure 7.5: Data copied to half pixel memory when horizontal and vertical propagation are used. Each row/column on the diagram is 4 pixels wide

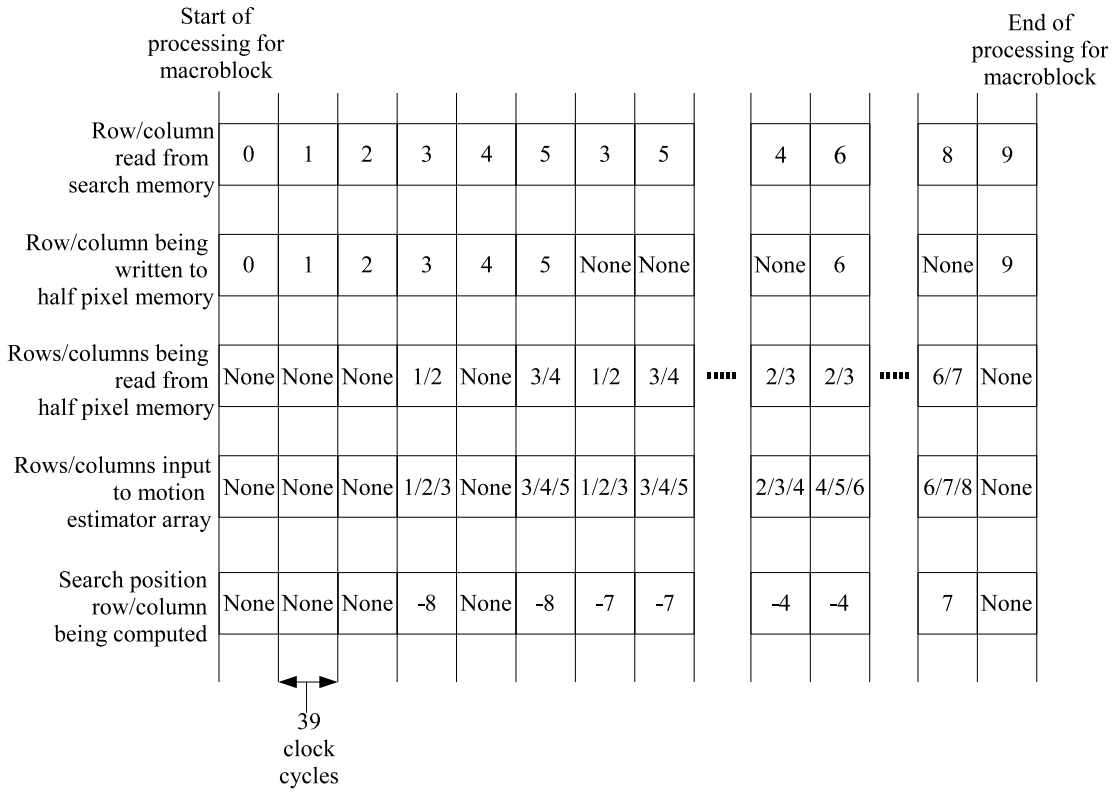

Figure 7.6: Timing of full search motion estimator

motion estimator. Reusing this data increases the initial latency of the motion estimator. The motion estimator has to wait until the required data has been read from the search memory into half pixel memory. After this initial period data from the search memory is copied into the half pixel memory when it is required by the motion estimation array. The last row/column is copied after the full pixel motion estimation operation has finished. This is illustrated in Figure 7.6.

The reference frame data outputted from the search memory and half pixel memory is aligned on a 4 pixel boundary. To be able to select the required data for the particular search position row or column being computed a multiplexer is used. With the appropriate 64 bit data being inputted into the motion estimation

array as shown in Figure 7.3.

7.3.3 Motion estimation unit

The motion estimation unit uses an array based architecture. To support variable block sizes individual 4x4 arrays are used to calculate the 4x4 SAD values, with the SADs for the large block sizes being summed from them as shown in in Figure 7.7. As discussed in section 2.3.2, this is the most common way in which variable block sizes are supported within a full search motion estimation architecture. As the input bit width is only 64 bits, only eight 4x4 arrays are used. As a consequence to calculate all 41 sub-block SADs for a particular search position two passes through the array are required. The first pass for the first eight 4x4 sub-blocks. The second pass for the last eight 4x4 sub-blocks as indicated in Figure 7.8.

To minimise the amount of time the motion estimator is inactive, the SADs for the first eight 4x4 sub-blocks for a complete search position row or column are calculated first. The SADs for the last eight 4x4 sub-blocks for the same search position row or column are then calculated. Using only eight 4x4 arrays reduces the LUT resources required by the estimator array whilst still providing enough processing resource for the motion estimator to complete its operation within 1400 clock cycles. However, to be able to calculate the SADs of block sizes which cross the first/second phase boundary, additional storage is required. Specifically the thirty-two 8x8 SAD values calculated in the first phase must be stored. When horizontal propagation is used, this enables the SAD values for the 16x16 and 8x16 block sizes to be calculated. When vertical propagation is used, this enables the SAD values for the 16x16 and 16x8 block sizes to be calculated. All other block sizes do not have sub-blocks which cross the first/second phase

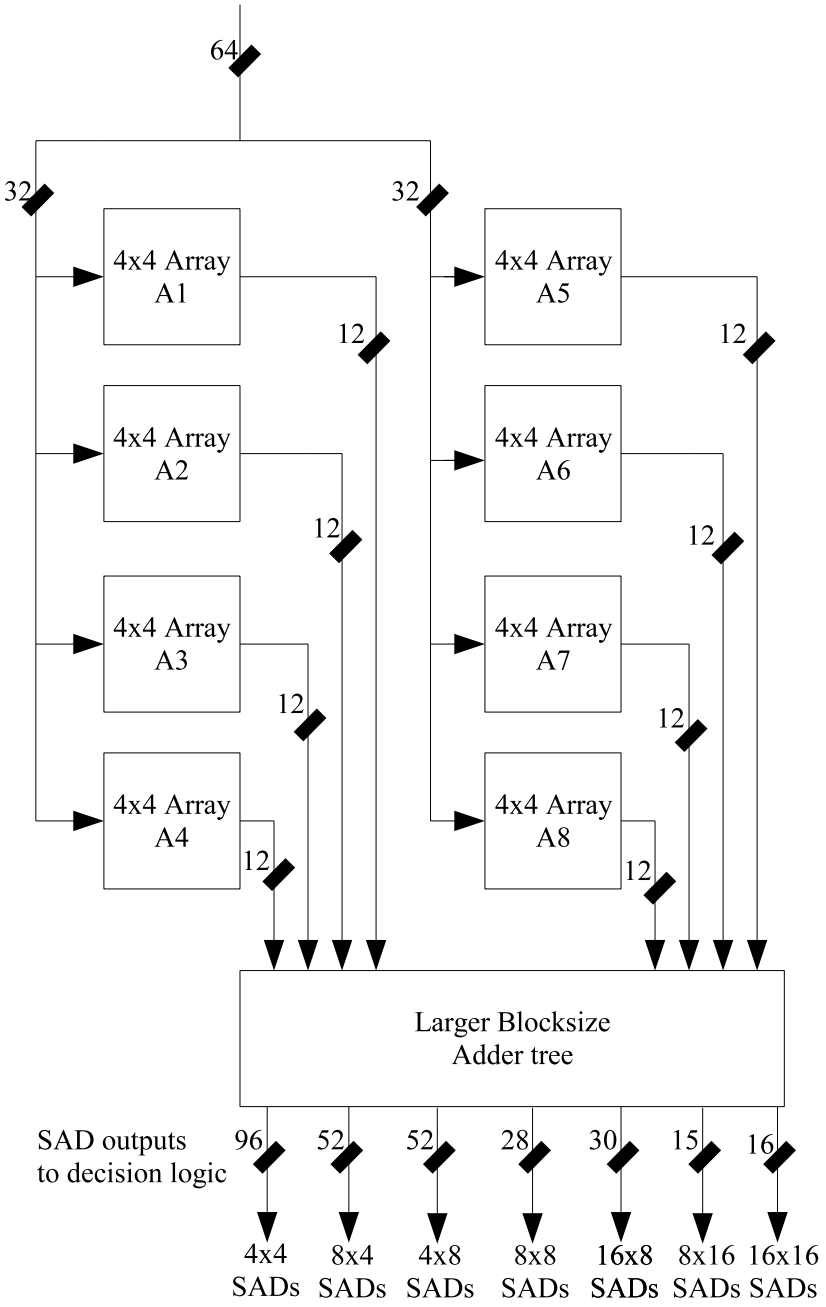

Figure 7.7: Full search array used

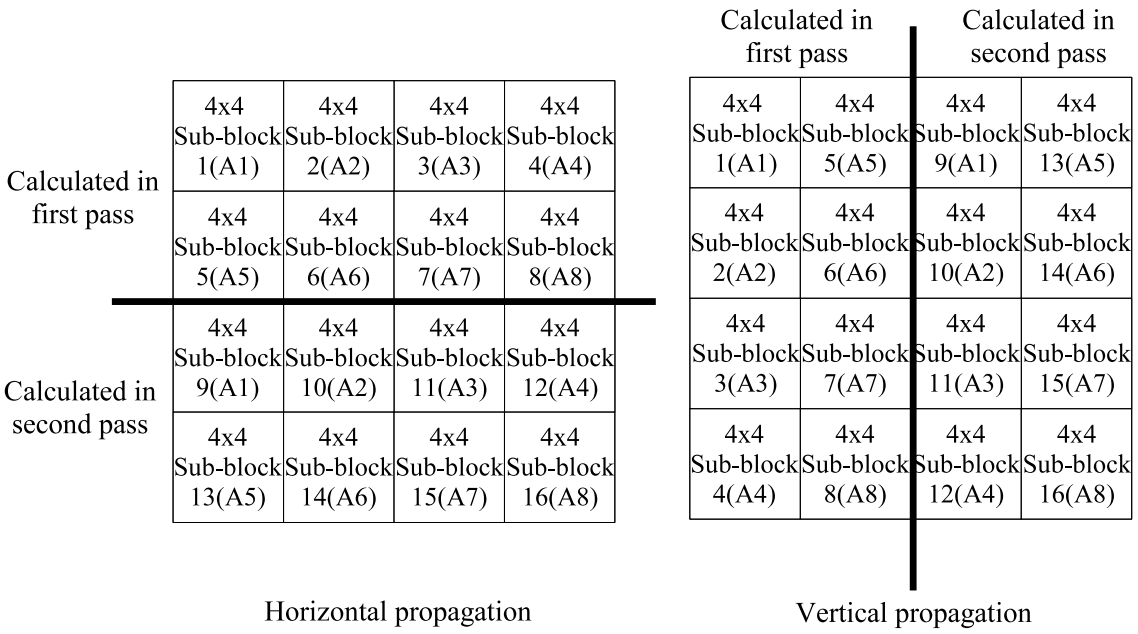

Figure 7.8: 4x4 sub-block SADs calculated in the first and second pass and the 4x4 array used for their calculation

boundary. Thus, the SAD values for these sub-blocks can be calculated as soon as all the SAD values for all the smaller block sizes required become available. In the current design, the SAD values required for larger block sizes become available at different clock cycles. It is therefore necessary to delay them by an appropriate number of clock cycles to allow the SAD values for larger block sizes to be calculated. More detailed timing information is given in appendix B.

Each 4x4 array is made up of 16 processing elements. A diagram of a single processing element is shown in Figure 7.9. Two current pixel registers are used, one to store the current pixel used in the first pass, the other to store the current pixel used in the second pass. Current macroblock data could be loaded into each 4x4 array as required, as opposed to only being loaded into the array at the start of the motion estimation process. This would remove the need for the second current pixel register and the 2 to 1 multiplexer. However, by using the

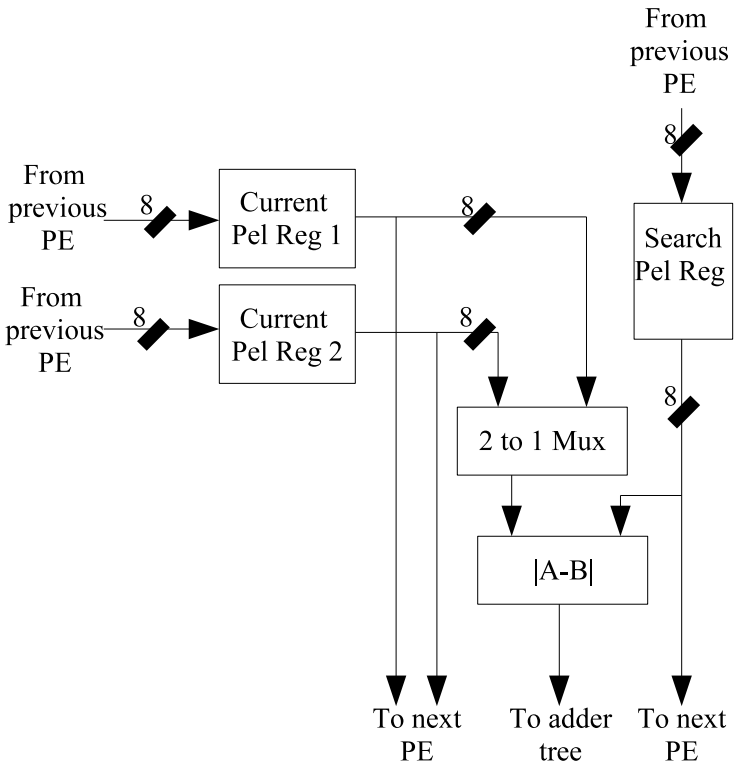

Figure 7.9: Processing element (PE) used in 4x4 arrays

additional register and multiplexer the input memory bandwidth is dramatically reduced. An adder tree is used to accumulate the 16 absolute difference values to produce the 4x4 SAD outputs.

7.3.4 Decision logic unit

The decision logic unit adds the appropriate rate term (section 2.3.1) to the SAD values produced by the motion estimation array. It then updates the best vector and SAD registers when a search position with a lower SAD value than the current best is found. Although there are 41 sub-blocks, only 20 adders/comparators are used in the decision logic unit. Implementing the motion estimation operation using two passes allows some of the adders/comparators used to be shared

between different sub-blocks.

In common with other full search variable block size implementations the rate term is calculated using only the predicted vector for the 16x16 block size. As discussed in section 2.3.2 all the vectors required to calculate the predicted vectors for other block sizes will not be available until the full pixel motion estimation process has finished.

7.4 Fractional Pixel Estimation Architecture

A diagram of the fractional search architecture is shown in Figure 7.10. The architecture implements the full fractional search algorithm described in section 2.3.3. The architecture used allows both the half and quarter pixel estimation operations required to be performed concurrently. When the half pixel estimator is determining the best half pixel vector for a sub-block, the quarter pixel estimator is determining the best quarter pixel vector for a sub-block whose best half pixel vector has previously been determined.

The complex half pixel interpolation process is performed once. This requires that the half pixel samples, as well as the necessary full pixel samples, are stored and available for use by the quarter pixel interpolator. A non trivial amount of resources are required to store the half pixel samples. However, in an FPGA it is more power efficient than performing the half pixel interpolation process twice. Embedded RAMs are used to store the half and full pixel samples because for the large amount of storage required, registers, as used in [19], would be inefficient.

To reduce the number of embedded RAMs used the interpolation and fraction estimation processes are not split across two pipeline stages, as was done previously. When supporting variable block sizes this would require over sixteen

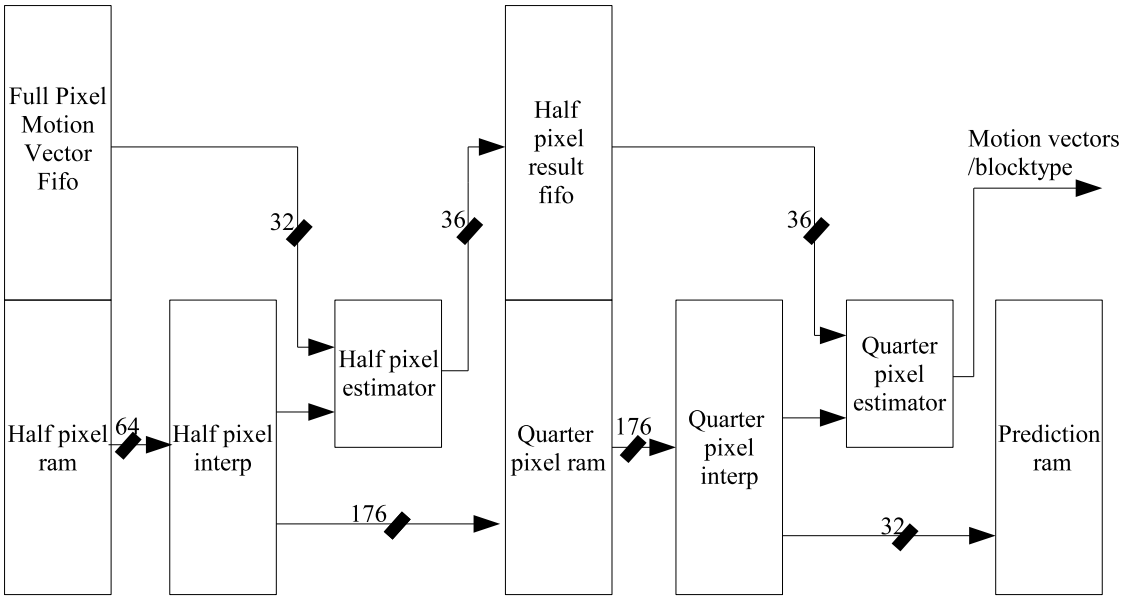

Figure 7.10: Fractional search architecture

1024 byte embedded RAMs to store all the full and half pixel samples for each sub-block, assuming double buffering was used. Not splitting the interpolation and fractional estimation processes across pipeline stages allows the quarter pixel memory to use only six 1024 byte embedded RAMs. However, not using multiple pipeline stages places does place greater demands on the performance of the half pixel interpolator.

7.4.1 Dataflow

Similarly to other fractional estimation implementations, the architecture is designed to operate primarily on sub-blocks with a block size of 4x4. Other block sizes are supported through decomposition into their constituent 4x4 parts. To support adaptive propagation without compromising hardware efficiency, 4x4 block integration takes place in the same direction data is propagated. This allows more of the full pixel samples required for the half pixel interpolation pro-

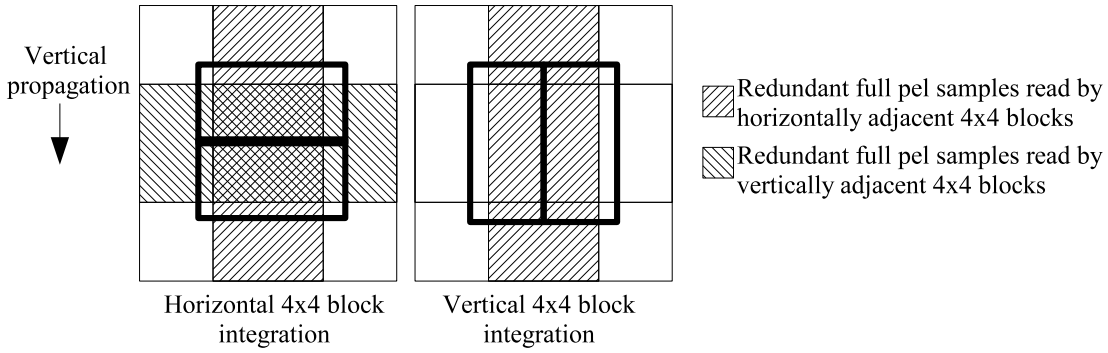

Figure 7.11: Redundant half pixel interpolation areas for a block size of 8x8 when vertical and horizontal integration are used, assuming data is propagated in the vertical direction

cess to be reused across constituent 4x4 blocks as shown in Figure 7.11. This reduces the half pixel memory bandwidth required. More importantly, it ensures that less clock cycles are spent filling the half and quarter pixel pipelines, increasing hardware utilisation. The architecture considers each block type in series. The 16x16, 16x8, 8x16 block sizes are considered first, then all the possibilities for each 8x8 sub-block (8x8, 8x4, 4x8 and 4x4). The precise processing order is dependent on the propagation direction.

Half and full pixel data is copied into the quarter pixel memory in vertical or horizontal strips which are 11 pixels wide. The length of each strip is dependent on the block size being processed. Each strip is 11 pixels to allow the quarter pixel interpolator to calculate any of the quarter pixel samples that may be required for the 4x4 block being considered. The exact quarter pixel samples required are not known when the full and half pixel data is being copied into the quarter pixel memory because the best half pixel vector is still being determined at this point.

As well as performing the fractional estimation process for each block type, the fractional estimation architecture must also ensure that the correct predicted block is copied to the prediction memory shown in Figure 7.10. This data needs to

be copied to enable the next pipeline stage to determine the difference block prior to the transform operation. The best block type and associated motion vectors will not be known until the end of the fractional motion estimation process. Therefore, the data which requires to be copied to the prediction memory will not be known until this time. The size of the quarter pixel memory used prevents the half pixel samples for all sub blocks being stored in it until the fractional estimation process has finished. Consequently, to generate the required data at this time, both the half and quarter pixel interpolation operations would need be repeated for the block type chosen. This is not done currently. Instead two 16x16 predicted blocks are copied to the prediction memory. One is the best predicted block for the 16x16, 16x8 and 8x16 block sizes. The other is the best predicted block for the 8x8, 8x4, 4x8 and 4x4 block sizes. For the 16x16, 16x8, and 8x16 block sizes the appropriate data is generated and copied into the prediction memory at the end of processing for that block size, if its cost is lower than the current best. For each 8x8 sub-block the appropriate data is copied into the prediction memory after all the block sizes for the 8x8 sub-block have been considered. This prevents multiple copying operations occurring for each 8x8 sub-block. The same method is not used for the three 16x16 partitions because there is not enough capacity in the quarter pixel memory to store the half pixel samples for the 16x16, 16x8 and 8x16 block sizes.

To support the processing of 4x4 blocks, the half pixel interpolation unit requires a 10 pixel, 80 bits wide input. However, the output port width of the half pixel memory is only 8 pixels and it is aligned on a 4 pixel boundary. To provide the 10 pixel input, the interpolation unit requires a buffer is used to store multiple 5 pixel rows, or columns, of reference data read from the half pixel memory as shown in Figure 7.12. The exact number is dependent on the

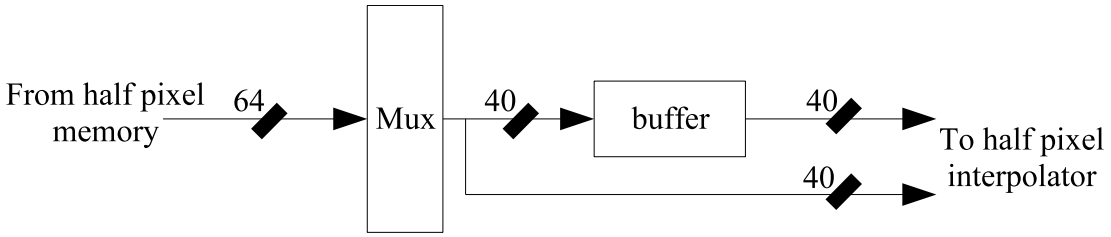

Figure 7.12: Buffer used to produce half pixel interpolator input

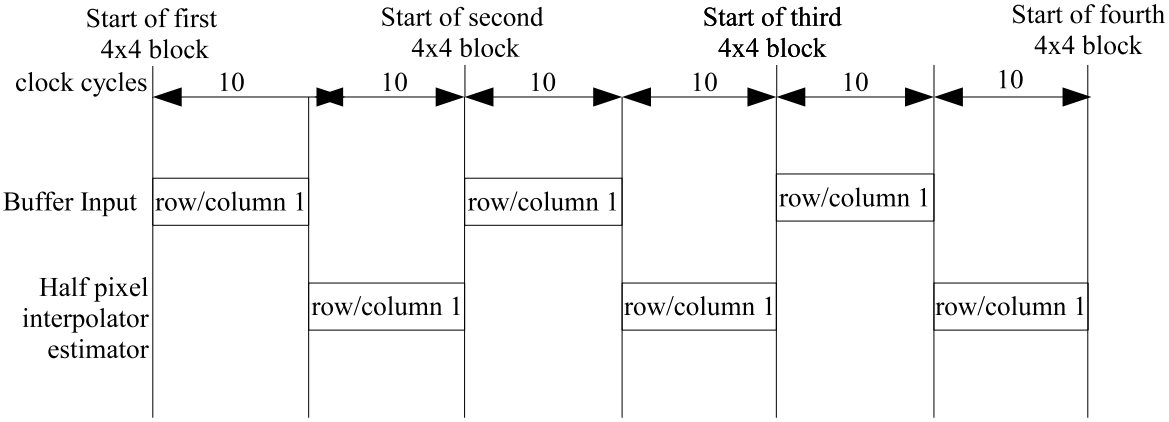

Figure 7.13: Timing of the half pixel interpolation/estimation operation for a block size of 4x4

block size of the sub-block being processed. The output of this buffer is then combined with data read directly from the half pixel memory to produce the 80 bit input, the half pixel interpolation unit requires. To minimize the number of clock cycles wasted loading the buffer, the data required for the next sub-block row, or column, is loaded into the buffer at the same time the half pixel interpolator and estimator are processing the current sub-block row, or column. This approach is not ideal. It reduces the achievable hardware utilisation to 50% for the 4x4 block size, as shown in Figure 7.13. For larger block sizes, such as 16x16, the impact on hardware utilisation is not as significant, with an 80% hardware utilisation achievable, as shown in Figure 7.14.

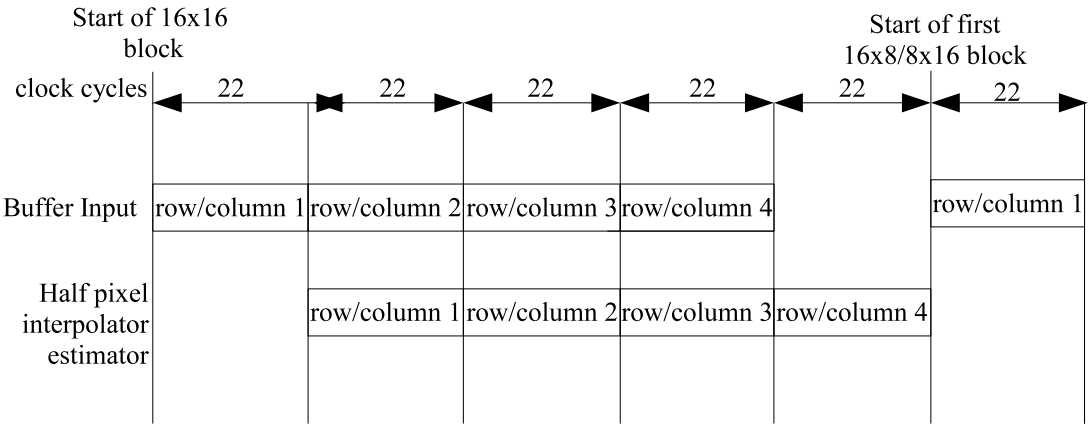

Figure 7.14: Timing of the half pixel interpolation/estimation operation for a block size of 16x16

7.4.2 Half Pixel Interpolator

The basic half pixel interpolation operation defined in the H.264 standard is given by

$$P_{hp} = A - 5B + 20C + 20D - 5E + F \tag{7.1}$$

when P_{hp} is the half pixel sample being generated and A, B, C, D, E and F are either full pixel samples or previously generated half pixel samples. Figure 7.15 is used to illustrated the correspondence between the location of samples A, B, C, D, E and F and the location of the generated sample P_{hp}. The horizontal half pixel samples labeled a, b, c, d, e and f in Figure 7.15 are given by,

a=A1-5*B1+20*C1+20*D1-5*E1+F1 d=A4-5*B4+20*C4+20*D4-5*E4+F4
b=A2-5*B2+20*C2+20*D2-5*E2+F2 e=A5-5*B5+20*C5+20*D5-5*E5+F5
c=A3-5*B3+20*C3+20*D3-5*E3+F3 f=A6-5*B6+20*C6+20*D6-5*E6+F6. (7.2)

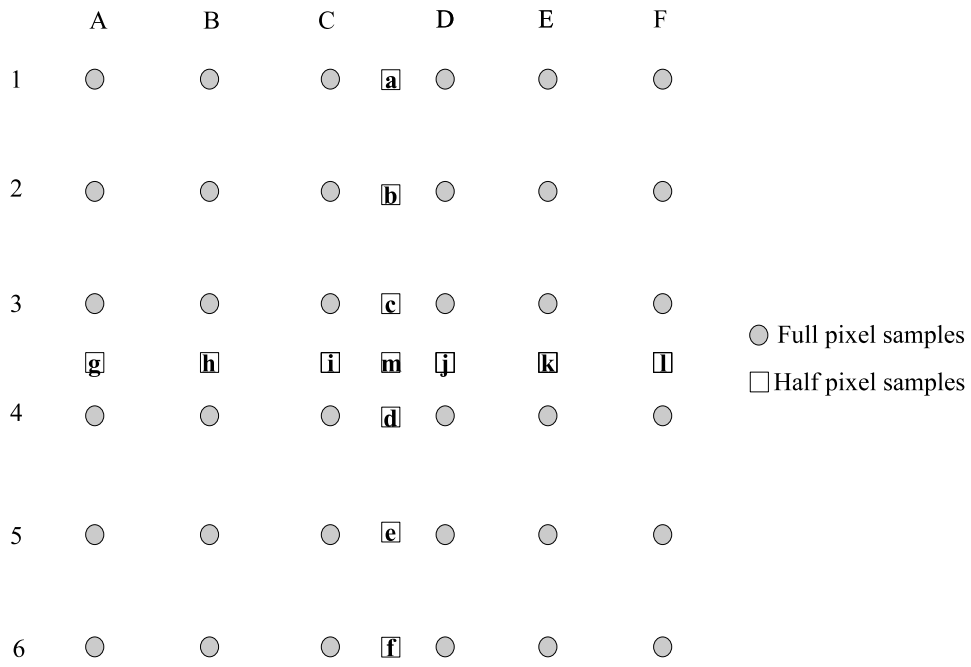

Figure 7.15: Correspondence between full pixel samples used and generate half pixel samples

The vertical half pixel samples labeled g, h, i, j, k and l in Figure 7.15 are given by,

$$\begin{array}{ll} \text{g=A1-5*A2+20*A3+20*A4-5*A5+A6} & \text{j=D1-5*D2+20*D3+20*D4-5*D5+D6} \\ \text{h=B1-5*B2+20*B3+20*B4-5*B5+B6} & \text{k=E1-5*E2+20*E3+20*E4-5*E5+E6} \\ \text{i=C1-5*C2+20*C3+20*C4-5*C5+C6} & \text{l=F1-5*F2+20*F3+20*F4-5*F5+F6.} \end{array} \tag{7.3}$$

Due to the symmetry of the interpolation operation the diagonal half pixel sample labeled m in Figure 7.15 is given by either

$$m = a - 5 * b + 20 * c + 20 * d - 5 * e + f \tag{7.4}$$

or

$$m = g - 5 * h + 20 * i + 20 * j - 5 * k + l. \tag{7.5}$$

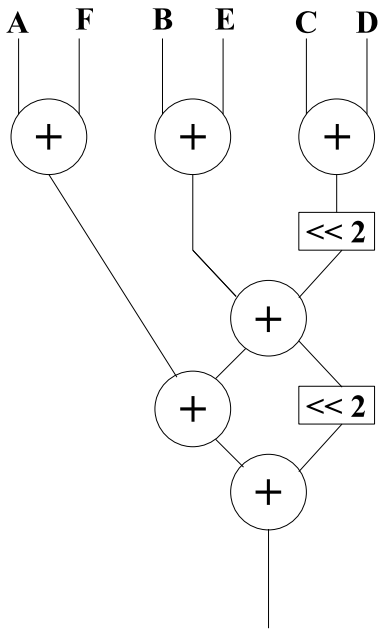

Figure 7.16: Basic interpolation unit used

The half pixel interpolation architecture used is similar to that described in [17]. The basic half pixel interpolation operation, given in equation (7.1), is implemented using 6 adders as shown in Figure 7.16. Although there are embedded multipliers available in the Cyclone-3 FPGA targeted, these are not used to implement the interpolation operation. Using an embedded multiplier would save at most one adder. Thus, it is unlikely any power saving would be realised from using them. It may be possible to implement the interpolation operation very efficiently using the embedded multiply accumulate units available on some high performance FPGAs. This has not been considered at present.

In total, twenty of the interpolation units shown in Figure 7.16 are used in the half pixel interpolator. The interpolation units are split across 3 filter banks as shown in Figure 7.17. Filter bank one generates, the vertical half pixel samples when horizontal propagation is used and the horizontal pixel samples when vertical propagation is used. Filter bank two generates, the horizontal half

pixel samples when horizontal propagation is used and the vertical half pixel samples when vertical propagation is used. Filter bank three always generates the diagonal half pixel samples, regardless of the propagation direction used. The structure takes advantage of the symmetry of the half pixel interpolation process to implement adaptive propagation. The diagonal half pixel samples being generated, from the horizontal half pixel samples as per equation 7.4 when horizontal propagation is used and from the vertical half pixel samples as per equation 7.5 when vertical propagation is used.

The structure of filter banks one and three is identical. Each uses five interpolation units (Figure 7.18) to generate the half pixels samples required by the half pixel estimator. Filter bank two uses ten interpolation units as shown in Figure 7.19. Only filter bank two outputs 3 to 6 are used by the half pixel estimator. The additional output samples are required to generate all the horizontal and vertical samples used as input to filter bank three. Output samples 2 and 6 of filter bank two are also saved to the quarter pixel memory because they may be needed by the quarter pixel interpolator.

7.4.3 Half Pixel Estimator

The half pixel estimator uses 8 processing elements of the type shown in Figure 7.20. Each processing element is used to determine the SAD value for one of the eight half pixel search positions. The centre full pixel cost is used as the initial best cost and position. It is available initially having already been determined by the full pixel motion estimator.

Only one comparator is used. Thus 8 clock cycles are required to determine the best half pixel vector and its associated cost. Currently the decision operation takes places sequentially, after the sum of absolute difference calculations. This

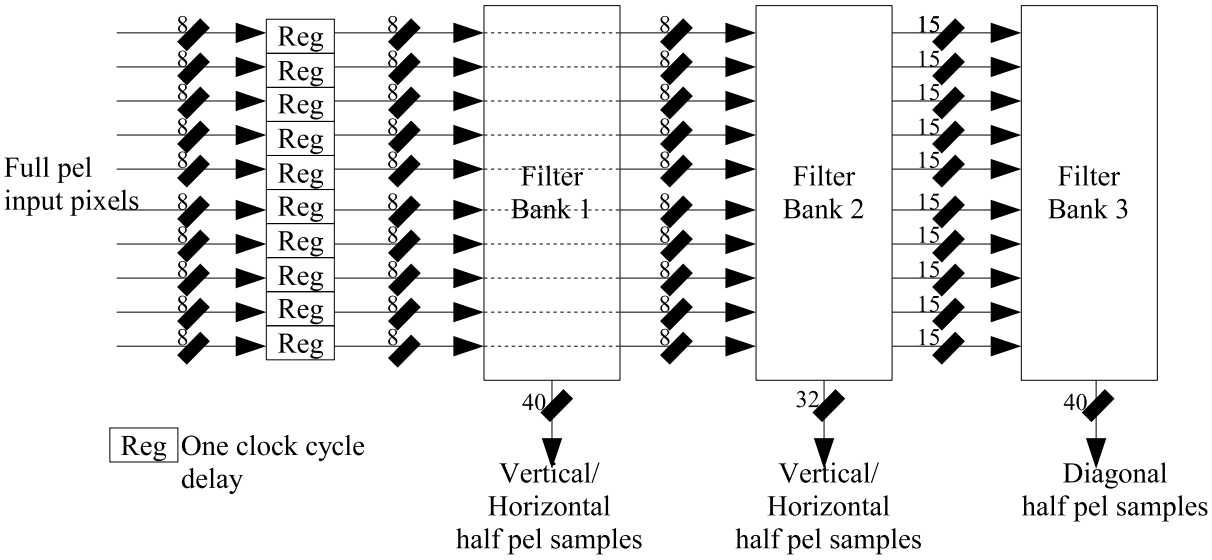

Figure 7.17: Overall filter structure

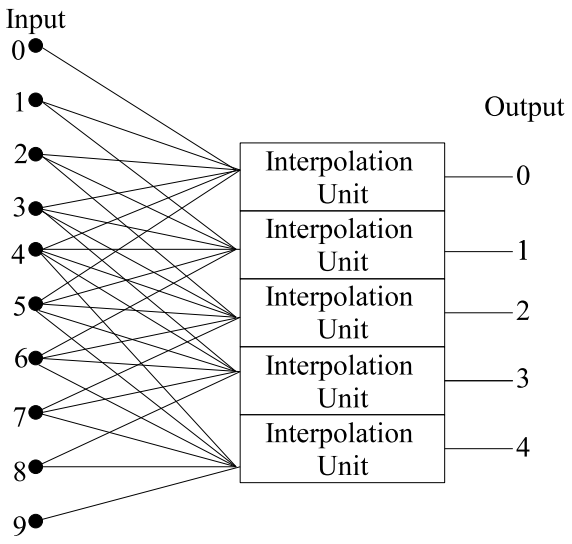

Figure 7.18: Structure of filter banks 1 and 3

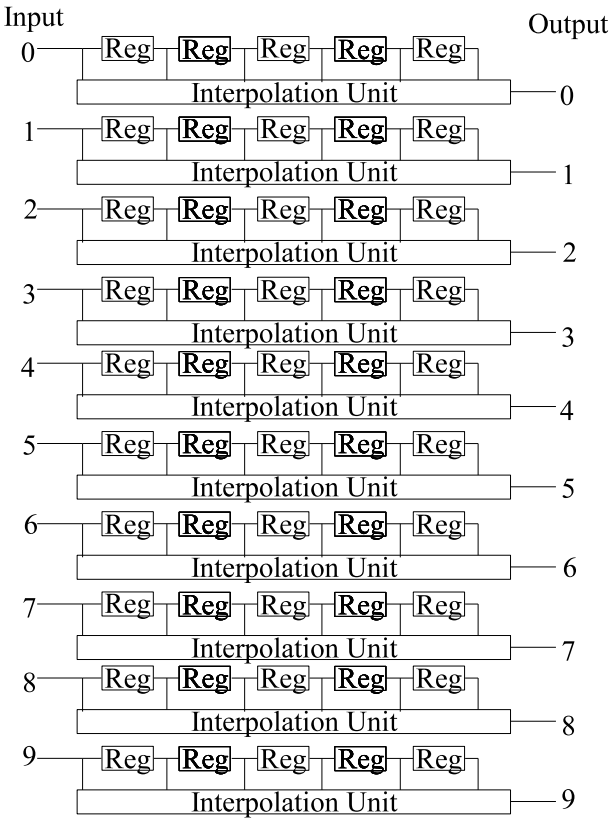

Figure 7.19: Structure of filter bank 2

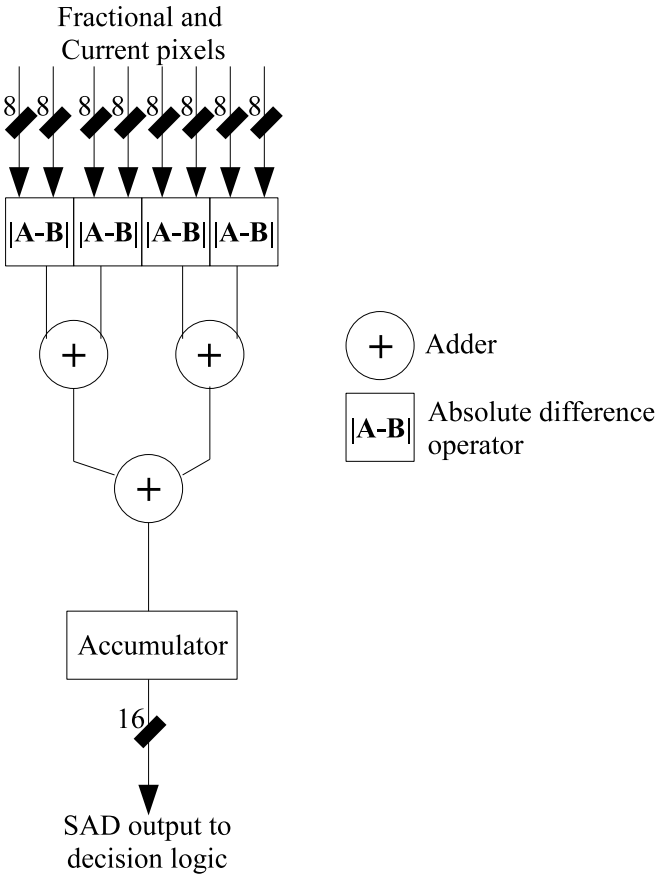

Figure 7.20: Structure of half pixel processing element

allows the decision logic to use the accumulator outputs directly. If additional registers were used it would be possible to pipeline these two operations, increasing logic utilisation. In the current design this would offer no benefit however. The buffer loading operation, discussed in section 7.4.1, always requires more than the 8 clock cycles required to determine the best half pixel vector.

7.4.4 Quarter Pixel Interpolator/Estimator

Data is saved to the half pixel memory in columns or rows of 11 pixels. Two adjacent rows or columns are read from the half pixel memory in a single clock cycle. This allows quarter pixel samples for all eight search positions to be calculate in the same clock cycle. It is necessary to read two rows or columns per clock cycle because of the minimal overlap between the full and half pixel samples required for the quarter pixel interpolation process. This is illustrated in Figure 7.21.

Multiplexers are used to select the data required by the quarter pixel interpolation units. For the horizontal and vertical quarter pixel samples this is a simple operation. The samples required are fixed in relation to the central half/full pixel sample. In addition to the search centre A, samples labeled B,C,D and E in Figure 7.22 are always required to calculate the horizontal and vertical quarter pixel samples. Selecting the half and full pixel samples for the diagonal quarter pixel locations is more complicated. The location of the required samples in relation to the central sample is dependent of the central sample location. If the centre sample is a full pixel sample, or diagonal half pixel sample, samples B,C,D and E are again required. If the centre sample is a horizontal or vertical half pixel sample, samples F,G,H and I are required. The interpolator units themselves are simple bi-linear filters, implemented using two adders and a fixed shift operation. The quarter pixel estimator is identical to the half pixel motion estimator

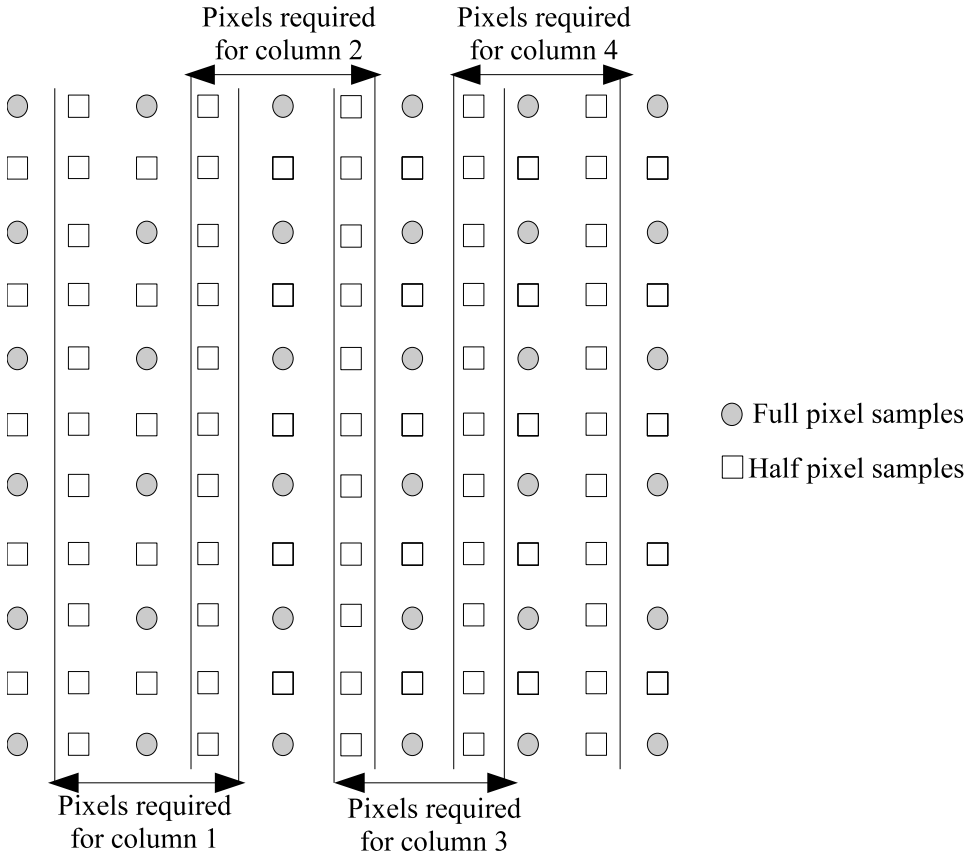

Figure 7.21: Full and half pixel samples required for 4x4 quarter pixel interpolation operation

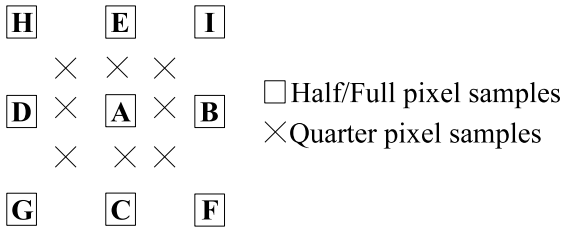

Figure 7.22: Location of full and half pixel samples required for quarter pixel interpolation

described in section 7.4.3.

7.5 Results and Discussion

7.5.1 Use of unencoded data for propagation decision

As discussed in section 7.1, for power efficiency reasons unencoded data is used to calculate the intra prediction results used by the propagation direction decision algorithm. To investigate the effect this has on the adaptive propagation algorithm's performance the Bit Accurate Model of the encoder studied was used. Figure 7.23 shows the percentage reduction achieved when the intra prediction results used were calculated using encoded and unencoded data. A quantisation parameter of 30 was used to encode the sequences.

From Figure 7.23, it can seen that the use of unencoded data makes negligible difference when a quantisation parameter of 30 is used. Similar results are achieved when lower quantisation parameters are used. When a quantisation parameter of 40 is used the differences are more noticeable as shown in Figure 7.24. This is not surprising. It would be expected that as the similarity between encoded and unencoded pixels decreases, the differences in performance would become more significant. Only in the case of two sequences, *table* and *outdoor*, is the performance of the adaptive propagation algorithm reduced sig-

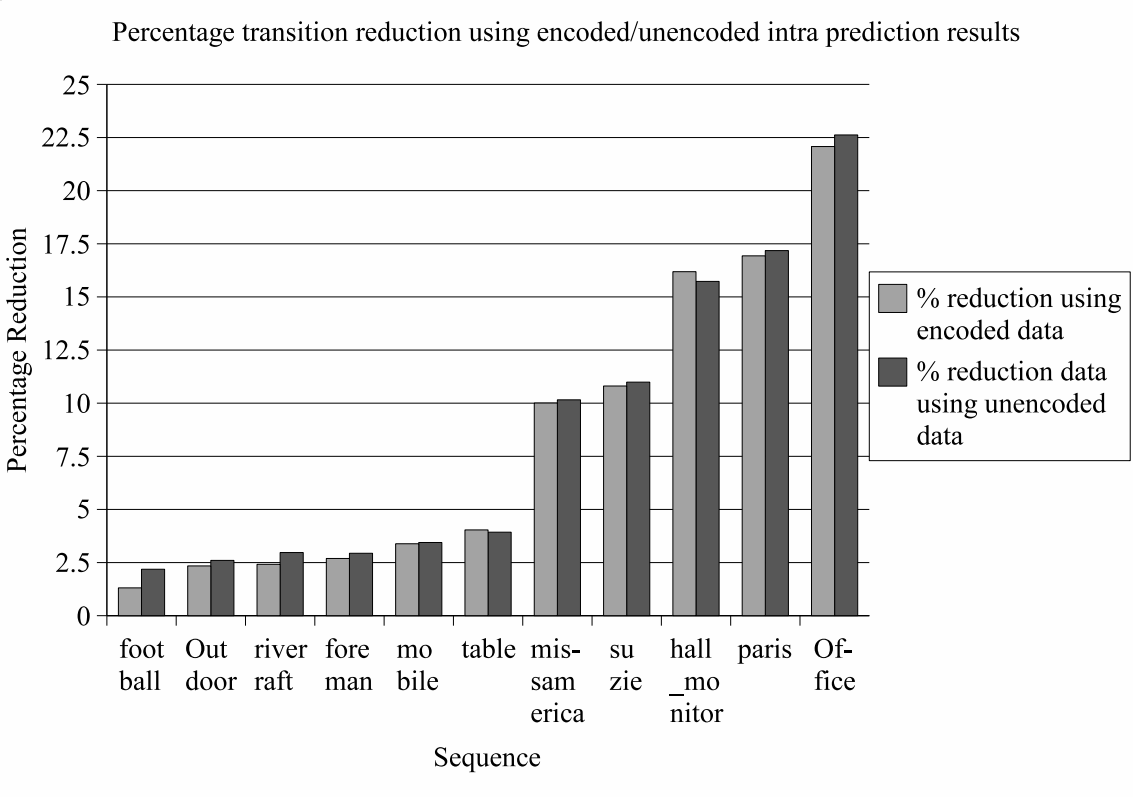

Figure 7.23: Percentage reduction in transitions when compared to horizontal propagation using intra prediction results calculated using unencoded and encoded pixels. A quantisation parameter of 30 was used to encode the sequences

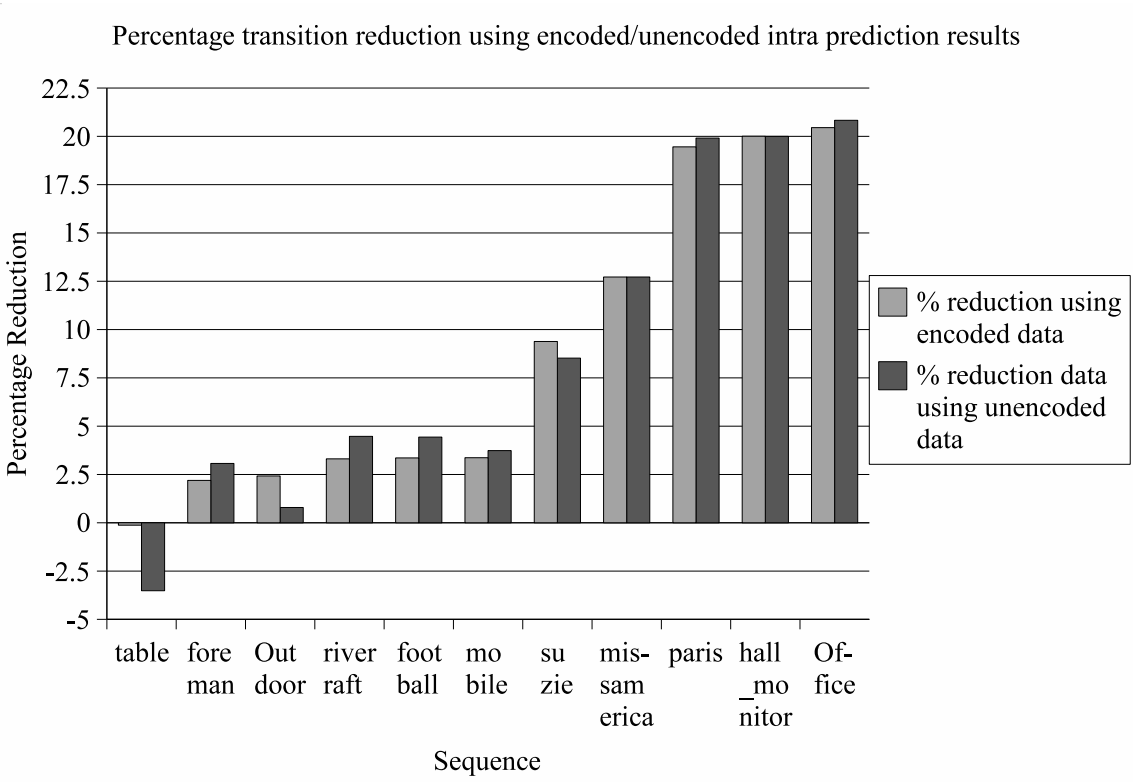

Figure 7.24: Percentage reduction in transitions when compared to horizontal propagation using intra prediction results calculated using unencoded and encoded pixels. A quantisation parameter of 40 was used to encode the sequences

nificantly by the use of unencoded data. Given this, and the advantages the use of unencoded data has in the encoder architecture studied, the use of unencoded data is justified.

7.5.2 Resources and Performance

Full Pixel Motion Estimator

The resources required by the motion estimation unit, decision logic unit and associated control logic when implemented in a Cyclone-3 FPGA are shown in table 7.1. Of note are the six M9K embedded RAMs used within the motion

Motion Estimation Unit	LUTs Registers M9K	6065 3777 6
Decision Logic Unit	LUTs Registers 9x9 Multipliers	2049 1047 1
Control Logic	LUTs Registers	373 178

Table 7.1: Full pixel motion estimator resource usage

estimation unit. Two of them are used to store the 8x8 SAD values that are required to be stored between the first and second passes for each search position row, or column. The other four are used to implement shift registers. These are needed to delay the smaller block SAD values used in the larger block SAD calculations. This is not ideal. It is an inefficient use of the embedded ram resource, given the relatively small amount of storage required. For FPGAs such as the Spartan-3, which can implement small shift registers very efficiently using LUTs, this inefficiency would not occur. For the Cyclone-3 FPGA used, the tree architecture (section 2.3.2) would be more appropriate as it does not require shift registers to delay the smaller block SAD values.

Fractional Pixel Motion Estimator

Table 7.2 shows the resources required by the fractional pixel motion estimator components. Six M9K embedded rams are required to store the half and full pixel samples needed by the quarter pixel interpolator. This is the only resource cost directly attributable to the decision to store the half and full pixel samples, instead of performing the half pixel interpolation operation twice. In principle, the same estimator could be used for the half and quarter pixel estimation operations, if the resources required needed to be reduced. Two estimators are used

Half Pixel Interpolator	LUTs Registers	2204 932
Half Pixel Estimator	LUTs Registers 9x9 Multipliers	1149 311 1
Quarter Pixel Interpolator and Estimator	LUTs Registers 9x9 Multipliers	2534 962 1
Quarter Pixel Ram	M9K	6
Half Pixel Result Fifo	M9K	1
Control Logic	LUTs Registers	552 239

Table 7.2: Fractional pixel motion estimator resource usage

currently to provide increased performance.

Currently the fractional pixel motion estimator takes between 1484 and 1700 clock cycles to complete the required operations for each macroblock. This is outwith the 1400 clock cycles per macroblock target. Substantial additional resources are not required for the fractional pixel motion estimator to meet the 1400 clock cycles per macroblock target. Instead a modification to the dataflow used is required. As discussed in section 7.4.1, currently a minimum of 2 and a maximum of 4 predicted blocks are copied to the prediction memory. If the dataflow is adjusted such that only the data actually needed is copied to the prediction memory, the architecture should be able to meet the 1400 clock cycles per macroblock target.

Resources Required to Support Adaptive Propagation

Table 7.3 shows the resource usage of the various modules which need to be modified to support the adaptive propagation algorithm. The main additional resource costs are incurred within the search memory unit and inter/intra mode decision unit. The search memory requiring an additional 100 LUTs and 9 registers to

		without adaptive propagation	with adaptive propagation
Search Memory	LUTs Registers	207 46	311 59
Intra/Inter Mode Decision	LUTs Registers	382 220	438 254
Direction Store	M9K	0	1
Output Ram	LUTs M9K Registers	5 2 1	8 2 1
Motion Estimator Input/Output	LUTs	0	40

Table 7.3: Resources required to support adaptive propagation algorithm

support the adaptive propagation algorithm. The intra/inter mode decision requiring 56 LUTs and 34 registers to perform the summation and comparison operations required by the adaptive propagation algorithm. In total the number of additional LUTs used represent less than 2% of those required by the motion estimation hardware, and even less if the encoder as a whole is considered. Similarly the registers and single embedded RAM needed to implement the adaptive propagation algorithm represent small fractions of those used by the complete encoder.

7.5.3 Power Used

To estimate the power used by both the full and fractional pixel motion estimators, and the power used implementing the adaptive propagation algorithm, the method documented in section 5.1.1 was used. To allow the impact of the adaptive propagation algorithm to be judged, the same full and fractional motion estimator netlist was simulated using both horizontal and adaptive propagation. This ensures that the impact of using adaptive propagation is not masked by any differences in synthesis, placement and routing. For the limited parts of

Size	Sequence	horizontal propagation (mW)	adaptive propagation (mW)	% Reduction with adaptive propagation
QCIF	Suzie	0.63	0.62	1.59
	Miss America	0.55	0.55	0
	Table	0.59	0.59	0
SIF	Football	2.14	2.14	0
CIF	Hall Monitor	2.28	2.25	1.32
	Paris	2.41	2.37	1.66
	Mobile	2.49	2.47	0.8
VGA	Office	6.67	6.6	1.05
	Riverraft	7.41	7.41	0
	Outdoor	6.75	6.73	0.3

Table 7.4: Power used by decision logic unit when adaptive propagation is and is not used

the design which require additional resources to support adaptive propagation this is not possible. For these parts of the design separate netlists were simulated. To initially test the performance of the adaptive propagation algorithm a quantisation parameter of 30 was used.

Full Pixel Motion Estimator

The power consumed by the full pixel control logic is constant, regardless of whether adaptive propagation is used or not. It consumes 0.14mW for a QCIF sequence, 0.48mW for a SIF sequence, 0.54mW for a CIF sequence and 1.68mW for a VGA sequence. The power consumed by the decision logic unit is reduced, by a small degree, by the use of adaptive propagation as shown table 7.4. As would be expected, given the results shown in chapter 6, the power reduction achieved is sequence dependent.

The majority of power consumed by the full pixel motion estimator is consumed within the motion estimation unit as shown in figure 7.25. The adaptive propagation algorithm has the greatest effect on this part of the full pixel motion

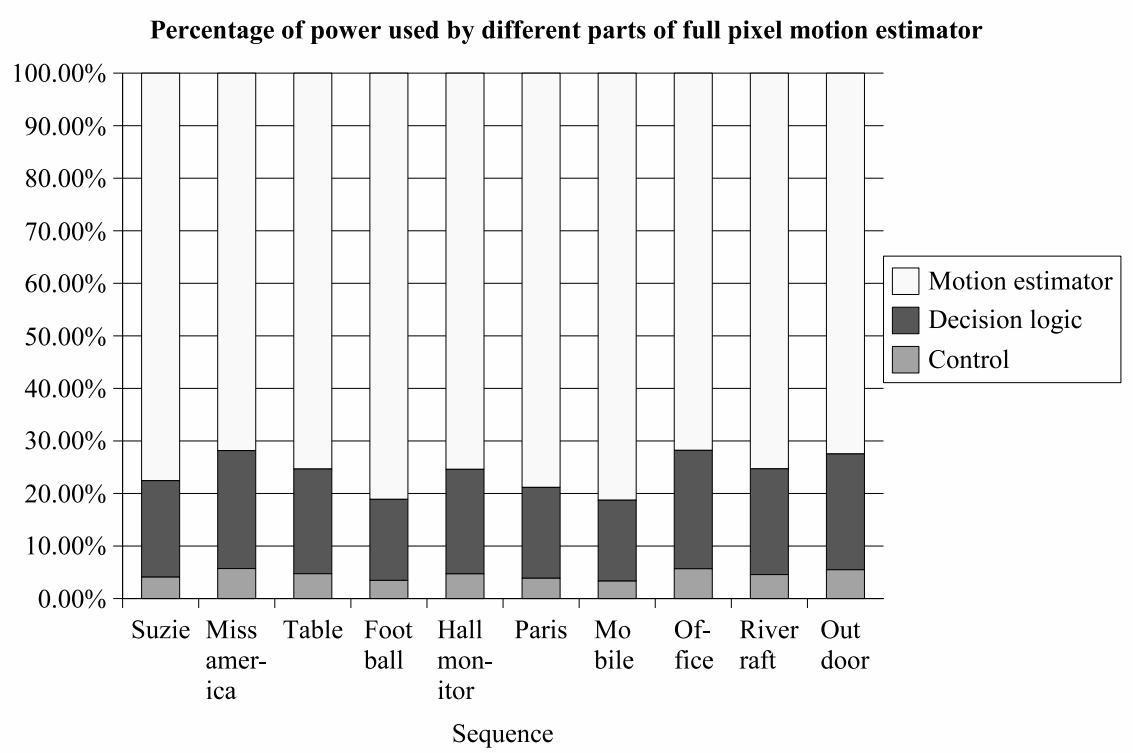

Figure 7.25: Percentage of power consumed by each section of full pixel motion estimation unit when adaptive propagation is not used

estimator. Table 7.5 shows the power used by the motion estimator unit when adaptive propagation is and is not used. The percentage power reduction achieved is less than the percentage transition reduction shown previously. This is principally a result of the algorithm only targeting one source of dynamic power, the power consumed routing data through the estimator. The power consumed by the registers and embedded RAMs as a result of them being clocked and enabled is not reduced by the algorithm. Nevertheless, as a result of the relatively high capacitance associated with FPGA interconnect, power reductions of between 7% and 10% can be achieved for sequences, such as *hall monitor*, *paris*, *suzie* and *office*, where the number of transitions is reduced by a significant amount.

For the *riveraft* and *football* sequences the power used by the motion estimator unit is actually increased when adaptive propagation is used. For both these sequences the adaptive propagation algorithm does not achieve a significant transition reduction. For other sequences with a similar percentage reduction in transitions using adaptive propagation still results in a small power reduction. The increase in power may be a result of the propagation direction decision not being appropriate for the entire search area. It is currently based on the intra prediction results of the macroblock at the centre of the search area. The search area used is larger than one macroblock. Thus, there is scope for the propagation direction decision to be incorrect.

As discussed in chapter 6, adaptively changing the propagation direction affects the most significant bits to a greater extent. It would be expected that the significance of any power saving would increase if bit truncation was used. However, the current full pixel motion estimation architecture is not designed to take advantage of this. The expense of supporting adaptive propagation must be incurred for all 8-bits regardless of the motion estimator unit's bit width. Full

Size	Sequence	horizontal propagation (mW)	adaptive propagation (mW)	% Reduction with adaptive propagation
QCIF	Suzie	2.66	2.46	7.52
	Miss America	1.76	1.71	2.84
	Table	2.23	2.22	0.45
SIF	Football	11.25	11.34	-0.8
CIF	Hall Monitor	8.64	7.76	10.19
	Paris	10.99	10.07	8.37
	Mobile	13.15	12.78	2.81
VGA	Office	21.24	19.66	7.44
	Riverraft	27.72	28.00	-1.01
	Outdoor	22.18	22.08	0.45

Table 7.5: Power used by motion estimation unit when adaptive propagation is and is not used. Sequences were encoded using a quantisation parameter of 30

8-bit data still needs to be copied to the half pixel memory.

Fractional Pixel Motion Estimator

The percentage of power consumed by the different parts of the fractional pixel motion estimator is shown in Figure 7.26. From Figure 7.26 it can be seen that the half pixel interpolator consumes the greatest amount of power. The power used storing the half and full pixel samples is approximately one fifth of the power required by the half pixel interpolation operation. Given this, it is clear that storing the half pixel samples offers a substantial power benefit. It saves approximately 80% of the power required for the second interpolation operation.

Whilst offering a power benefit, storing the half pixel samples reduces the impact of the adaptive propagation algorithm. As shown in Figure 7.27, the adaptive propagation algorithm reduces the power used by the half pixel interpolator to a greater extent than it reduces the power used by other parts of the fractional pixel motion estimator. If the half pixel interpolation operation was repeated, as in other architectures which have been proposed, the overall impact

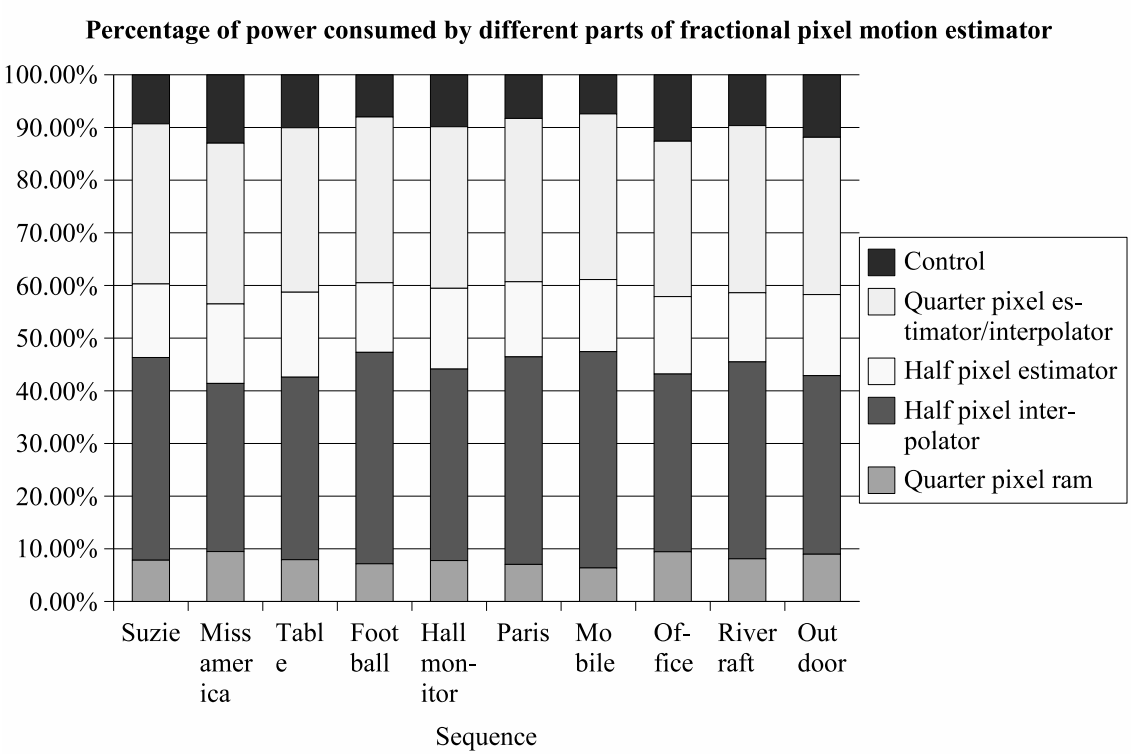

Figure 7.26: Percentage of power consumed by each part of fractional pixel motion estimator when adaptive propagation is not used

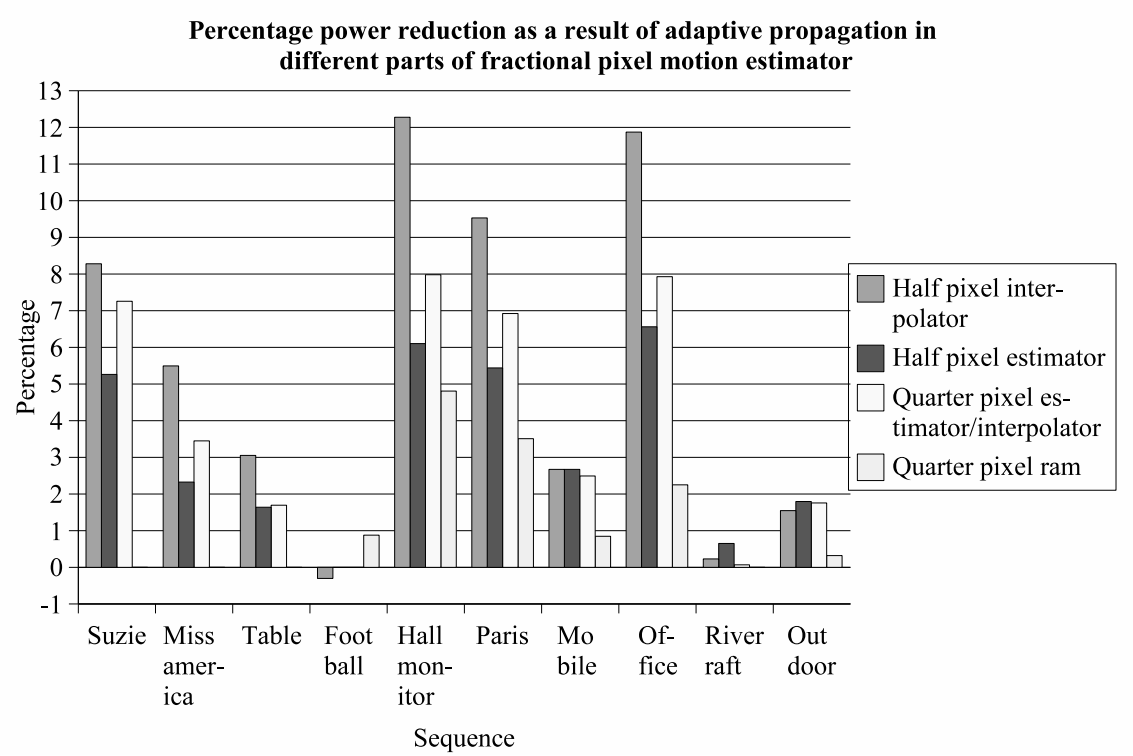

Figure 7.27: Percentage power reduction in different parts of fractional pixel motion estimator when adaptive propagation is used

adaptive propagation has on the fractional pixel motion estimator's power consumption would increase. Note that the control logic is not shown in Figure 7.27 because the adaptive propagation algorithm has negligible impact on the power consumed by it. Full details of the power consumption of the fractional pixel estimator are given in appendix E

The differing effect adaptive propagation has on the various components can be explained. The half pixel interpolation component operates solely on encoded pixels. As previously discussed the adaptive propagation algorithm has a greater impact on encoded data as a result of the least significant bit switching activity being reduced by the encoding process. The half pixel estimator, however, operates on both encoded reference frame, and unencoded input frame, pixels. The

Size	Sequence	Power used supporting adaptive propagation (mW)
QCIF	Suzie	0.06
	Miss America	0.05
	Table	0.06
SIF	Football	0.25
CIF	Hall Monitor	0.23
	Paris	0.25
	Mobile	0.26
VGA	Office	0.62
	Riverraft	0.78
	Outdoor	0.68

Table 7.6: Power required to support adaptive propagation algorithm. Sequences were encoded using a quantisation parameter of 30

adaptive propagation algorithm does not reduce the switching activity of unencoded data to the same degree it reduces the switching activity of encoded data. Therefore, the effect adaptive propagation has on the half pixel estimator's power consumption is less. The effect of the current pixel data's switching activity on the power consumption of the full pixel motion estimator is less pronounced because new current pixel data is only loaded into it at the start of the estimation process. Due to the use of 4x4 decomposition this is not possible in the fractional pixel motion estimator.

Power used supporting the adaptive propagation algorithm

The total power used supporting the adaptive propagation algorithm is shown in table 7.6. Over 90% of the power used supporting the algorithm is consumed within the search memory. The actual propagation direction decision consuming less than 0.01 milliwatts for each QCIF sequence and a maximum of 0.02 milliwatts for the riveraft sequence. This suggests that, in an encoder architecture which can tolerate a motion estimator with a relatively high latency, the cost of implementing the adaptive propagation algorithm will be minimal.

Size	Sequence	Horizontal propagation (mW)	Adaptive propagation (mW)	% Reduction with adaptive propagation
QCIF	Suzie	9.72	9.33	3.97
	Miss America	7.45	7.37	1.02
	Table	8.92	8.91	0.07
SIF	Football	38.06	38.45	-1.03
CIF	Hall Monitor	32.97	31.09	5.7
	Paris	38.49	36.57	4.98
	Mobile	43.26	42.68	1.34
VGA	Office	87.79	84.12	4.18
	Riverraft	108.13	109.16	-0.95
	Outdoor	90.86	90.95	-0.1

Table 7.7: Total power used by the full and fractional pixel motion estimators and the search memory when adaptive propagation is and is not used

7.5.4 Overall Power Savings

The total power used by the full and fractional pixel motion estimators and the search memory, when adaptive propagation is and is not supported, is shown in table 7.7. Note that the power consumed when the adaptive propagation algorithm is used includes all the other costs associated with supporting the adaptive propagation algorithm.

From table 7.7 it can be seen that the adaptive propagation algorithm can be implemented with minimal additional power consumption. Only in the case of three sequences do the costs associated with implementing adaptive propagation outweigh the internal FPGA power savings. For the *outdoor* sequence this is by a negligible amount. For the other two sequences, *riverraft* and *football*, supporting adaptive propagation incurs a power cost of 1 milliwatt and 0.38 milliwatts respectively.

Figures 7.28 and 7.29 plot the total power used, internally within the FPGA and on the memory bus itself, as a function of bus capacitance. Two sequences, *paris*

and *riverraft*, are used. Note that in figures 7.28 and 7.29 only the bus transitions caused by the luminance component of each sequence are taken into account. The power cost of using adaptive propagation with each reference frame's chrominance components has not been accounted for. Therefore, it would be unfair to consider the reduction in chroma component transitions the adaptive propagation algorithm provides. To calculate the power used communicating with external memory equation, 3.1 was again used. The power estimated derived from equation 3.1 was multiplied by four to reflect the fact that the level-C data reuse scheme and a vertical search range of 16 pixels has been assumed.

From Figure 7.28 it can be seen that, for the *paris* sequence, an overall reduction in power can be achieved using adaptive propagation. The addition of the memory bus power increases slightly the percentage power saving, with total power savings between 5 and 6 percent depending on bus capacitance and voltage. For the *riverraft* sequence the addition of the memory bus power does not significantly alter the percentage power increase adaptive propagation incurs using this sequence. This is a result of the adaptive propagation algorithm having minimal impact on the bus transitions which occur when encoding this sequence. A 1% power increase is incurred as a result of supporting adaptive propagation. The other sequences have results within this range. The influence of memory bus power consumption is less than that shown in chapter 5 because a lower bus voltage has been assumed. The motion estimator used is also more complex, resulting in the motion estimator itself consuming a larger proportion of the total power.

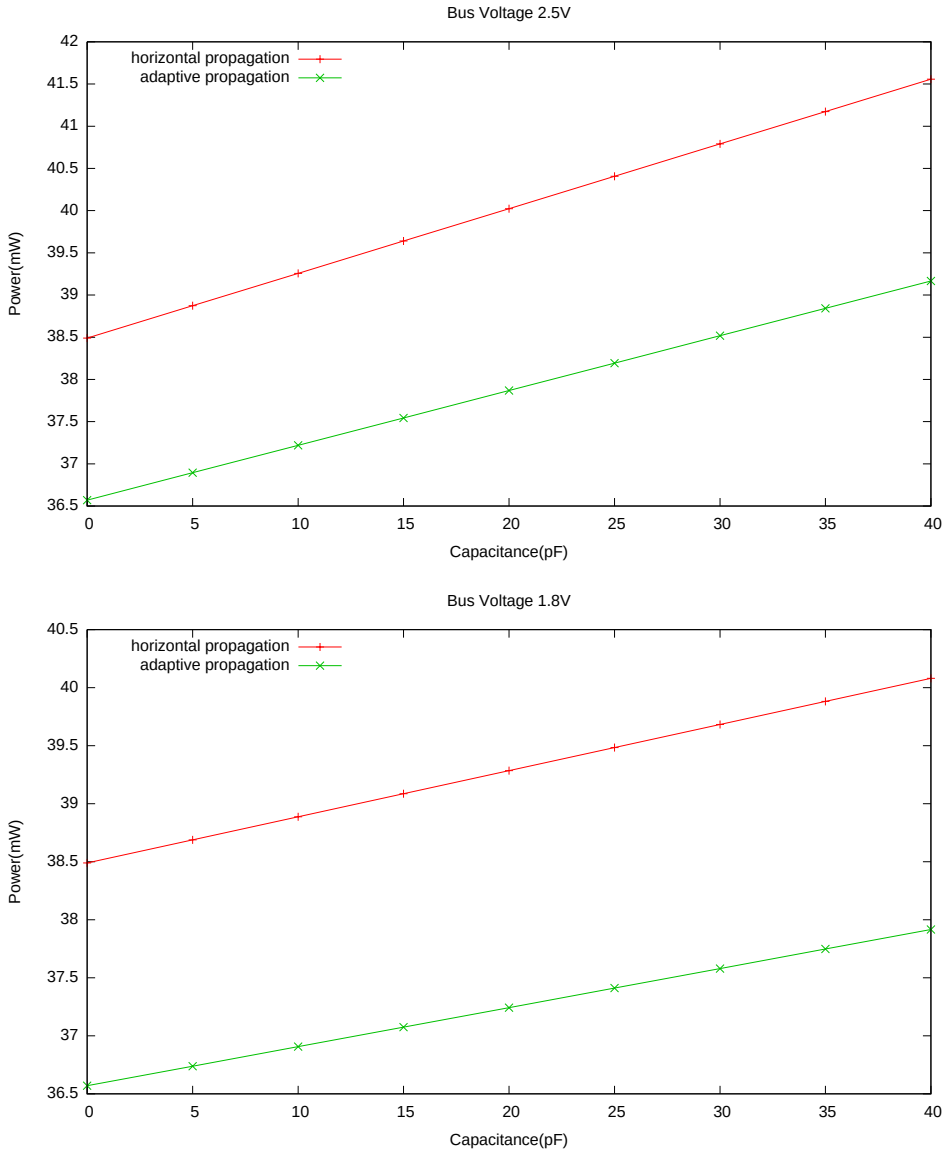

Figure 7.28: Total power used as a function of bus capacitance for a 2.5 and 1.8 volt bus - paris sequence

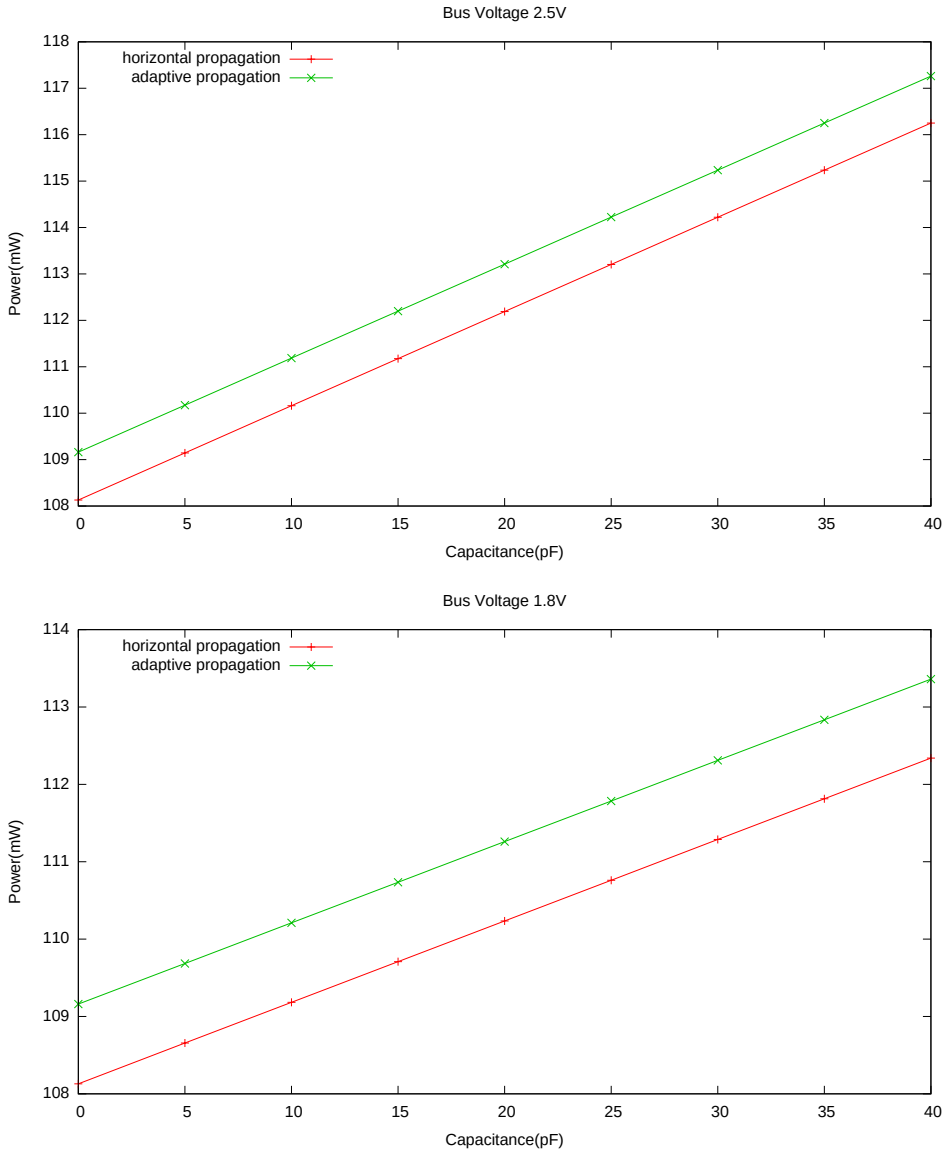

Figure 7.29: Total power used as a function of bus capacitance for a 2.5 and 1.8 volt bus - riverraft sequence

7.6 Summary

The implementation of the adaptive propagation algorithm into a pipelined encoder has been presented. Power can also be reduced in the motion estimation stage of a video encoder using the adaptive propagation algorithm. The adaptive propagation algorithm reducing power by up to 13% for individual parts of the motion estimator. However, as the adaptive propagation algorithm only reduces the power caused by data transitions, an overall power reduction of up-to 6% only is achieved. More importantly, for sequences where the adaptive propagation algorithm does not reduce transitions significantly, the power used is actually increased by up to 1%.

Chapter 8

Conclusions and Further Work

8.1 Conclusions

This thesis has explored the implementation of video encoding algorithms on reconfigurable devices. This work has specifically targeted FPGAs, currently the most widely used reconfigurable logic devices. Two different areas were explored, FPGA video encoding systems and the power consumed by FPGA based video encoders.

Chapter 4 focused on FPGA video encoding systems. Two video encoding systems were implemented. The first video encoding system targeted video surveillance applications. The approach used to implement this system was to isolate as much as possible the encoding sub-system and the processor sub-system. While this simplified the overall system design it limited the size of the system software, and hence the features the overall system could support. As discussed in section 4.1.4 using the cachelink interface now supported by the microblaze processor may make it easier for the processor and encoding system to share a single memory. This would allow the system to support additional features. The sec-

ond system targeted video conferencing applications. In this system the FPGA had to implement less functionality than in the previous system as the encoder control and streaming functions were implemented on other devices. Due to the encoding requirements, however, the system design was still challenging. A custom architecture had to be used to allow the external memory to operate at the frequency required.

Chapter 5 documents the results of a power analysis performed on a pipelined video encoder when implemented in a Cyclone-2 FPGA. It was shown that for low resolution sequences (SIF/CIF) FPGA static power dominates. However for higher resolutions such as D1 the dynamic power consumption becomes more significant, accounting for more that half of the total FPGA power consumption. The significance of FPGA static power consumption is well known. The results do quantify the limited overall effect that reducing the dynamic power used would have, when encoding single low resolution sequences. They also show that for higher resolution sequences reducing dynamic power will, potentially, have a more significant effect. Whilst no results are directly provided, it can also be concluded that, when encoding multiple low resolution sequences, reducing dynamic power would also be beneficial.

The rest of the thesis focused on reducing dynamic power in an FPGA based video encoder. In the remainder of chapter 5 the motion estimation function was identified as the key source of dynamic power consumption in an H.264 FPGA video encoder. It was also shown that, configuring the search memory so that a minimum number of embedded RAMs are enabled per clock cycle reduces the power consumed by the search memory substantially. As a consequence, a much smaller proportion of the total power consumed by the motion estimator is consumed by the search memory compared to what occurs in ASIC motion

estimator implementations. In the results presented the log search algorithm was used. It would be expected that a similar power distribution would be observed with other search position reduction algorithms. Given this difference it would be worth comparing the total power consumed by different search algorithms when implemented in FPGAs, similar to the investigation conducted for ASICs reported in [53].

In chapters 6 and 7 an algorithm, and associated architecture, to reduce the amount of dynamic power used loading and saving reference frame data from external memory and within the motion estimator computational units was proposed. The basis of the algorithm is that by changing the order data is propagated, the number of data transitions, and hence the dynamic power used, can be reduced. A unique feature of the algorithm is that it uses results generated during the encoding process. Specifically, the 4x4 intra prediction results are used to determine the propagation order. This is beneficial because it reduces the costs associated with implementing the algorithm.

In chapter 6 it was shown that external memory bus transitions and hence the power used loading and saving reference frame data is reduced by nearly 20% when the algorithm is used. In addition the algorithm also works in conjunction with other bus encoding algorithms such as bus invert and DBM/VBM. Given this it may also be of use in ASIC video encoder implementations.

In chapter 7 an FPGA architecture to implement the algorithm is developed. A key implementation decision was to apply the algorithm in the motion estimation stages of the video encoder, in addition to applying it to the reference frame data written to and read from external memory. Results showed this to be beneficial. For the majority of sequences the algorithm reduced the dynamic power in both the full and fractional pixel motion estimation stages. However,

for two sequences results indicated that the proposed algorithm actually increases power consumption, albeit slightly.

The main advantages of the proposed power reduction algorithm are its relatively low resource costs and its wide applicability. It reduces power consumption in a number of video encoding stages. However, the algorithms usefulness is inhibited by its sequence dependence. In chapter 6 it was shown that whilst a transition reduction of nearly 20% is obtained for some sequences, for others a transition reduction of less than 5% is obtained. In chapter 7 it was shown that for two of the sequences tested, the algorithm caused a slight increase in power consumption overall. Further development of the algorithm/architecture used could potentially reduce this disadvantage.

Also in chapter 7, a novel FPGA architecture for the full fractional motion estimation algorithm was developed. It is shown that, by storing the half pixel samples in embedded RAM prior to the quarter pixel search operation, less power is consumed. This is a beneficial contribution with respect to implementing the full fractional search algorithm, However, using a 1 step fractional motion estimation algorithm, which does not have separate half and quarter pixel estimation stages, may offer a greater power reduction. Further research is required to quantify the power reduction and compression performance achievable using reduced complexity fractional motion estimation algorithms.

8.2 Further Work

In addition to the future work discussed in the preceding section, a number of other potential directions for further research have been identified,

- Physical FPGA power measurements. These would allow the estimation

methodology used in this thesis to be further verified and enable a more comprehensive set of power results to be generated for the proposed adaptive propagation algorithm.

- Effect of pipelining in combination with the proposed adaptive propagation algorithm. Given that pipelining reduces the power used in the FPGA routing network (by reducing glitches), it is probable that the benefit of using the proposed adaptive propagation algorithm will be reduced as the level of pipelining used by the full and fractional pixel motion estimators is increased.

- Adaptively changing the encoder clock frequency. Throughout this thesis it has been assumed that the number clock cycles per pipeline stage C_p, and hence the clock frequency required for a specific frame rate and resolution, is fixed. Clock distribution consumes a substantial amount of power in any FPGA design. Designing a pipelined encoder to use a variable C_p, would allow the encoder clock frequency to be adjusted, depending on the characteristics of the sequence being encoded. This could potential offer a significant power reduction. However, to realise such an encoder the algorithms and architectures used by each pipeline stage would need to be designed carefully.

- Considering the use of reconfigurability within a pipelined encoder. In this thesis the ability to reconfigure an FPGA in response to a change in its environment has not been considered. This represents a significant area for further research. For example the encoding modes supported by the encoder could be modified depending on the type of sequence being encoded.

Appendix A

H.264 Stereo Video Compression

In this appendix the use of the H.264 standard to compress stereo video sequences is studied. This work was undertaken with the view to implementing a H.264 stereo video encoder in a FPGA. The research was not pursued, however, because results indicated that exploiting the additional redundancy present in stereo video compression offered little benefit.

A.1 Stereo Video

Humans perceive the world in three-dimensions, despite the fact that the eyes only ever provide access to two-dimensional images of the world; a three-dimensional perception, therefore, must be constructed from the two-dimensional images received. Humans, more specifically the Human Visual System, do this using a number of depth cues including linear perspective, object overlapping, shading, accommodation, convergence, texture gradient, motion parallax, and disparity. The importance to depth perception of each depth cue varies with distance. One of the most important, at short viewing distances, is disparity. Disparity is the

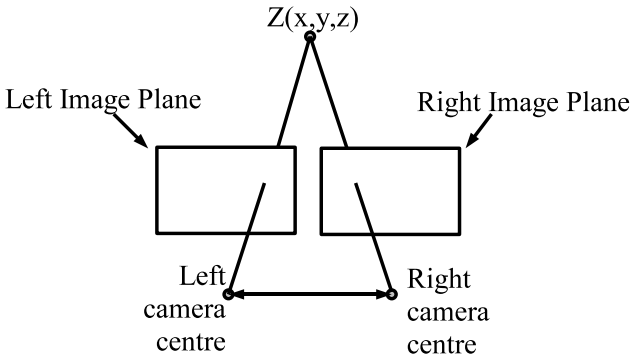

Figure A.1: Parallel camera configuration used to capture a stereo video sequence

difference between the position of an object in the images received by the left and right eye. It is a binocular depth cue requiring both eyes to function correctly

Ordinary television screens and computer monitors do not re-create the disparity depth cue. The same image is presented to each eye. To successfully recreate the disparity depth cue different images need to be presented to each eye. In general a parallel camera configuration is used to capture stereo sequences as shown in figure A.1. This produces the most comfortable images for the viewer. For stereo image display a number of techniques have been developed such as anaglyph - where colour is used to separate the left and right eye images; and polarisation - where a light's polarisation is used to separate the left and right eye images [131].

In terms of applications, the most popular current use is in tele-operation. Tele-operation systems using stereo video have been employed in industrial [132], medical, and military applications. It has been shown that for a number of remote operation tasks stereo video provides a benefit; with stereo video the user typically requires less training to perform the remote operation task correctly [133]. The other successful current application is 3D cinema. The recent increase in popularity of 3D cinema being due to the use of better display technologies,

which minimise viewer discomfort; and the production of high quality content. A larger research effort has been conducted into 3D-television. Numerous European 3D-TV research projects have been initiated, including ATTEST[1] and more recently the 3D-TV Network Of Excellence[2]. It is unlikely, however, that 3D television will be launched commercially in the short term.

In addition to temporal and spatial redundancy, stereo video sequences also have binocular redundancy. This extra redundancy potentially allows greater compression performance. In this appendix the use of H.264 for stereo video compression is explored, with a view to implementing an FPGA based stereo video encoder.

A.2 Previous Work

A.2.1 Stereo Video Compression using H.264

The multi-view image compression problem was first considered in [134]. In [134] it was proposed to use fixed sized block matching to exploit the redundancy between the images within a multi view image set. This is not unusual. The majority of the techniques applicable to mono video compression are also applicable to stereo video compression. As a result the improvement in the performance of stereo video compression algorithms has tracked the performance improvement in monocular compression algorithms.

There are three generic structures for stereo video encoding, simulcast, compatible, and joint as shown in figure A.2. To aid the exploitation of binocular redundancy disparity estimation is used, in a similar way to how motion esti-

[1]Refer to http://www.hitech-projects.com/euprojects/attest/summary.htm
[2]Refer to https://www.3dtv-research.org/

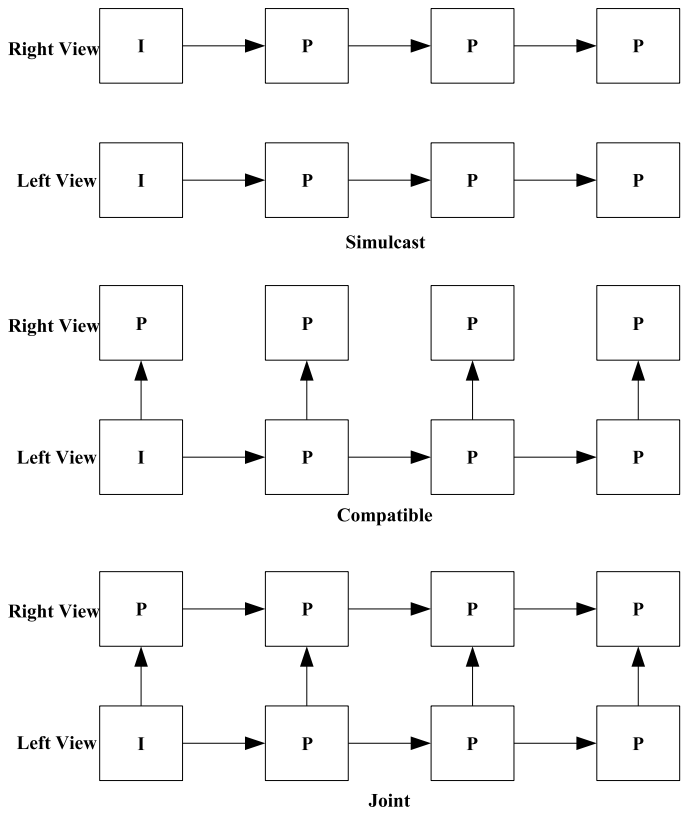

Figure A.2: Structures for stereo video compression. Horizontal arrows indicate motion estimation, vertical arrows indicate disparity estimation

mation aids the exploitation of temporal redundancy. Only the joint structure exploits all the temporal, spatial, and binocular redundancies present in a stereo video sequence

A fourth type of redundancy has been suggested in [135]. Wordline redundancy is the redundancy between different views at different time instances. While it undoubtedly exists, only [131] has quantified the improvement. Results given in [131] indicate that for H.264 stereo video exploiting wordline correlation in addition to the other redundancy types provides a performance improvement of between 0.1 and 0.2 db.

In [131] the joint encoding structure, and variations which supported wordline

correlation were implemented using the multiple reference frame option present in the H.264 standard. In this type of H.264 stereo implementation, left and right images are processed in turn by a single H.264 encoder. The produces a single stream which can be decoded by a standard H.264 decoder. In order for the H.264 decoder to know which images belong to which view it requires additional information.

The format of this additional information was standardised, as part of the H.264 supplementary enhancement information (SEI) syntax. In addition to supporting the buffer management stereo method the stereo SEI also supports the use of the H.264 interlaced coding options for stereo video compression. Each field in a picture being one of the images in a stereo pair [136]. Using interlaced video to support stereo has been used with analogue video standards such as PAL and NTSC. This implementation method also supports all the redundancies present in a stereo sequence, assuming the H264 field encoding mode is used. Surprisingly in [136] results were produced showing that when the macroblock adaptive frame field option was enabled, the compression performance was slightly better that when each field was compressed separately.

A.2.2 Illumination Compensation

All block matching methods make the implicit assumption that an object's intensity will be consistent between frames being compared. With regard to images displaced in time this assumption generally holds, although there are some sequence specific instances when it does not [137]. For spatially separated images any sequence can suffer from illumination mismatch, either as a result of differences within the cameras used, or as a result of different amounts of light being reflected into each camera. In stereo video the illumination differences are

generally attributable to camera mismatches as the difference in the two camera's positions is small. To improve compression it is desirable to remove any illumination differences between the left and right view before a stereo pair is compressed.

In [138] histogram modification is used in order to reduce the illumination differences between the two images in a stereo pair. A less complex method used by [139][140][141] is to model the illumination difference between two corresponding points in each image using a scaling and an offset,

$$S_r(x, y) = aS_l(x, y) + b. \tag{A.1}$$

where a and b are constants and S_r and S_l are the luminance values at corresponding points in the left and right images respectively. The values of a and b are determined such that the first and second order moments of the left and right image are equal. The corrected luminance values for the whole right image are then produced using

$$S_{rc}(x, y) = aS_r(x, y) + b \tag{A.2}$$

A.3 Experiments

The simulcast methods offers a number of implementation advantages compared to the joint method. The processing requirement is reduced. More significantly, with simulcast there is not data dependency between the left and right views. This allows the left and right views to be compressed concurrently, making it easier to implement a low latency encoder. Low latency is an important requirement for all tele-operation applications.

Encoding experiments were performed to determine whether the compression benefit offered by the joint encoding method, outweighed the implementation advantages of the simulcast method. The compatible method was not considered because for the majority of macroblocks temporal, instead of binocular, redundancy is favored. Thus, the compatible method provides poorer compression performance than simulcast, while offering no implementation advantages.

For the simulcast structure only a single temporally adjacent frame was used for reference. For the joint structure only a single temporal and single binocular frame was used for reference. Other more complicated encoding structures have not been considered. The H.264 JM 9.0 reference software was modified to support stereo operation [130]. Stereo video was loaded on an alternating left/right pattern, similar to the manner reported in other H.264 stereoscopic encoding proposals. The reference list management options within H.264 were utilised in order to ensure that the motion reference frame was situated in reference 0, list 0, for both simulcast and joint encoding structures. This improves the performance of the joint structure by ensuring that using motion reference frame requires the least number of encoding bits. The other modification made was to introduce a specific function for disparity estimation. This differed from the motion estimation function in that it was strongly horizontal biased, reflecting the fact that disparity, when the parallel camera configuration is used, only has a horizontal component.

Three stereo video sequences were used. One of the sequences used *diplo* reflects a common tele-operation application, remote vehicle operation (refer to table A.1) To correct any illumination differences present between the left and right views the simple gain/offset model, given in equation A.2, was used. The a and b coefficients were determined by equating the first and second order mo-

.

Name	Dimensions	Description
antonio	176x256	Large Main Object movement
diplo	320x240	Fast Camera/Object movement
talking	320x240	Minimal Main object Movement, Some background movement

Table A.1: Images used in experiments

ments. To ensure no intra-view illumination differences the a and b co-efficients were only calculated for the first frame in each sequence and then applied to all frames in the sequence.

A.4 Results and Discussion

To compare the quality of the encoded images the PSNR metric was used. We only consider the right view of the stereo sequence, as the left sequence is processed identically in all cases. Figure A.3 shows the PSNR curves of the right frame for the joint and simulcast structures. Both the *antonio* and *diplo* sequences show a compression performance improvement of approximately 0.3 dB when the joint structure is used. The *talking* sequence shows a performance improvement of less than 0.1 dB.

The *talking* sequence shows noticeable illumination differences between the left and right view. If illumination compensation is applied to the right view of the *talking* sequence the compression performance of the joint stereo structure increases. With illumination compensation, the joint structure provides a near 0.3 dB performance improvement over the simulcast method. For the *antonio* and *diplo* sequences the use of illumination compensation would appear to decrease the compression performance of the joint-p structure. This may be due

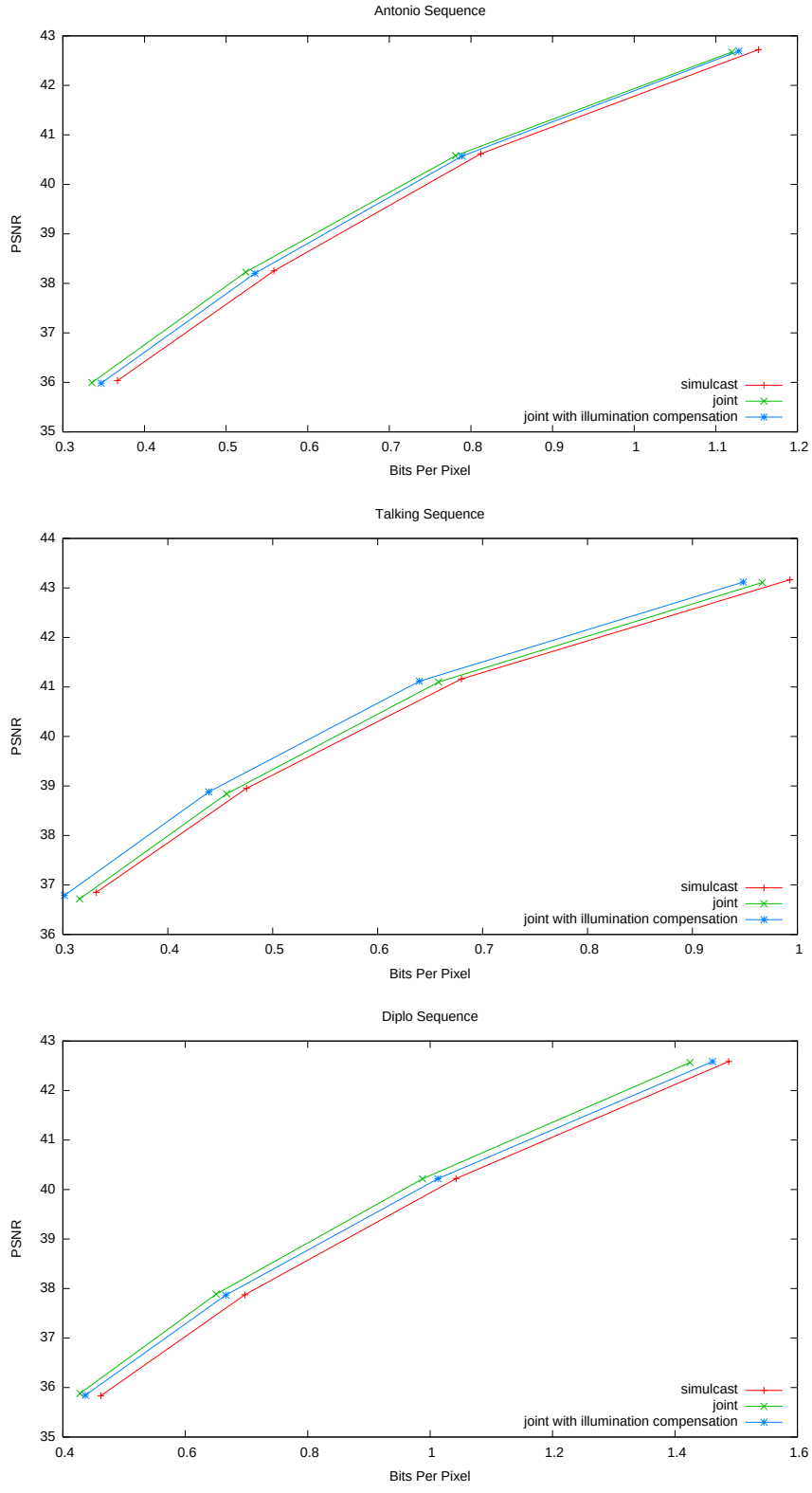

Figure A.3: PSNR curves for antonio, talking and diplo sequences when the simulcast and joint encoding methods were used

to the calculated a and b co-efficients from the first frame not being accurate for future frames.

These results presented are similar to the results reported in [136] [142]. Although [142] did provide results indicating that for a sequence with a large number of scene changes the joint method provides a performance improvement of nearly 0.8 dB. In [131] results are given indicating that 1 dB performance improvement is achievable using the joint method for certain sequences.

The two most popular current applications of stereo video are 3D cinema and tele-operation. For the cinema application, intra frame compression techniques are predominant to meet the picture quality requirement of that application. For the tele-operation application there will be no scene changes. Therefore the performance improvement offered by the joint method does not outweigh the implementation disadvantages. In particular the increased latency required to implement the joint method is likely to be unacceptable.

A.5 Conclusion

Exploiting the binocular redundancy present in stereo video sequences offers a compression benefit. However the improvement provided is not enough to justify pursuing an FPGA implementation, given the requirements of the applications which currently make use of stereo video.

Appendix B

Full Pixel Motion Estimation Array Timing

In this appendix timing information for the full search motion estimator described in chapter 7 is given. Only information for the first search position row/column is provided. The timing relationships for the other search position rows/columns are identical. Table B.1 provides details of the input pixel and the corresponding 4x4 SAD output timing. For table B.1 the origin for the reference pixel indices is assumed to be the upper left corner of the search area. Table B.2 shows the timing of the SAD outputs for the 4x8 and 8x4 block sizes. Table B.3 shows the timing of the SAD outputs for the 8x8, 16x8, 8x16 and 16x16 block sizes. For tables B.2 and B.3 it has been assumed that horizontal propagation has been used. If vertical propagation was used the timing of the SAD outputs for the 8x4 and 4x8 block sizes would be swapped, as would the timing of the SAD outputs for 16x8 and 8x16 block sizes.

Clock Cycle	Reference Pixel Input ((X,Y) or (Y,X) dependent on propagation direction)	Search Position Output ((X,Y) or (Y,X) dependent on propagation direction)			
		A1/A5	A2/A6	A3/A7	A4/A8
0	(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)(0,7)				
1	(1,0)(1,1)(1,2)(1,3)(1,4)(1,5)(1,6)(1,7)				
2	(2,0)(2,1)(2,2)(2,3)(2,4)(2,5)(2,6)(2,7)				
3	(3,0)(3,1)(3,2)(3,3)(3,4)(3,5)(3,6)(3,7)				
4	(4,0)(4,1)(4,2)(4,3)(4,4)(4,5)(4,6)(4,7)	(-8,-8)			
5	(5,0)(5,1)(5,2)(5,3)(5,4)(5,5)(5,6)(5,7)	(-8,-7)			
6	(6,0)(6,1)(6,2)(6,3)(6,4)(6,5)(6,6)(6,7)	(-8,-6)			
7	(7,0)(7,1)(7,2)(7,3)(7,4)(7,5)(7,6)(7,7)	(-8,-5)			
8	(8,0)(8,1)(8,2)(8,3)(8,4)(8,5)(8,6)(8,7)	(-8,-4)	(-8,-8)		
9	(9,0)(9,1)(9,2)(9,3)(9,4)(9,5)(9,6)(9,7)	(-8,-3)	(-8,-7)		
10	(10,0)(10,1)(10,2)(10,3)(10,4)(10,5)(10,6)(10,7)	(-8,-2)	(-8,-6)		
11	(11,0)(11,1)(11,2)(11,3)(11,4)(11,5)(11,6)(11,7)	(-8,-1)	(-8,-5)		
12	(12,0)(12,1)(12,2)(12,3)(12,4)(12,5)(12,6)(12,7)	(-8,-0)	(-8,-4)	(-8,-8)	
13	(13,0)(13,1)(13,2)(13,3)(13,4)(13,5)(13,6)(13,7)	(-8,1)	(-8,-3)	(-8,-7)	
14	(14,0)(14,1)(14,2)(14,3)(14,4)(14,5)(14,6)(14,7)	(-8,2)	(-8,-2)	(-8,-6)	
15	(15,0)(15,1)(15,2)(15,3)(15,4)(15,5)(15,6)(15,7)	(-8,3)	(-8,-1)	(-8,-5)	
16	(16,0)(16,1)(16,2)(16,3)(16,4)(16,5)(16,6)(16,7)	(-8,4)	(-8,-0)	(-8,-4)	(-8,-8)
17	(17,0)(17,1)(17,2)(17,3)(17,4)(17,5)(17,6)(17,7)	(-8,5)	(-8,1)	(-8,-3)	(-8,-7)
18	(18,0)(18,1)(18,2)(18,3)(18,4)(18,5)(18,6)(18,7)	(-8,6)	(-8,2)	(-8,-2)	(-8,-6)
19	(19,0)(19,1)(19,2)(19,3)(19,4)(19,5)(19,6)(19,7)	(-8,7)	(-8,3)	(-8,-1)	(-8,-5)
20	(20,0)(20,1)(20,2)(20,3)(20,4)(20,5)(20,6)(20,7)		(-8,4)	(-8,-0)	(-8,-4)
21	(21,0)(21,1)(21,2)(21,3)(21,4)(21,5)(21,6)(21,7)		(-8,5)	(-8,1)	(-8,-3)
22	(22,0)(22,1)(22,2)(22,3)(22,4)(22,5)(22,6)(22,7)		(-8,6)	(-8,2)	(-8,-2)
23	(23,0)(23,1)(23,2)(23,3)(23,4)(23,5)(23,6)(23,7)		(-8,7)	(-8,3)	(-8,-1)
24	(24,0)(24,1)(24,2)(24,3)(24,4)(24,5)(24,6)(24,7)			(-8,4)	(-8,-0)
25	(25,0)(25,1)(25,2)(25,3)(25,4)(25,5)(25,6)(25,7)			(-8,5)	(-8,1)
26	(26,0)(26,1)(26,2)(26,3)(26,4)(26,5)(26,6)(26,7)			(-8,6)	(-8,2)
27	(27,0)(27,1)(27,2)(27,3)(27,4)(27,5)(27,6)(27,7)			(-8,7)	(-8,3)
28	(28,0)(28,1)(28,2)(28,3)(28,4)(28,5)(28,6)(28,7)				(-8,4)
29	(29,0)(29,1)(29,2)(29,3)(29,4)(29,5)(29,6)(29,7)				(-8,5)
30	(30,0)(30,1)(30,2)(30,3)(30,4)(30,5)(30,6)(30,7)				(-8,6)
Start of Second Pass					
31	(0,8)(0,9)(0,10)(0,11)(0,12)(0,13)(0,14)(0,15)				(-8,7)
32	(1,8)(1,9)(1,10)(1,11)(1,12)(1,13)(1,14)(1,15)				
33	(2,8)(2,9)(2,10)(2,11)(2,12)(2,13)(2,14)(2,15)				
33	(3,8)(3,9)(3,10)(3,11)(3,12)(3,13)(3,14)(3,15)				
33	(4,8)(4,9)(4,10)(4,11)(4,12)(4,13)(4,14)(4,15)	(-8,-8)			
60	(29,0)(29,1)(29,2)(29,3)(29,4)(29,5)(29,6)(29,7)				(-8,5)
61	(30,0)(30,1)(30,2)(30,3)(30,4)(30,5)(30,6)(30,7)				(-8,6)
Start of second search position row/column					
62	(0,0)(0,1)(0,2)(0,3)(0,4)(0,5)(0,6)(0,7)				(-8,7)
63	(1,0)(1,1)(1,2)(1,3)(1,4)(1,5)(1,6)(1,7)				
63	(2,0)(2,1)(2,2)(2,3)(2,4)(2,5)(2,6)(2,7)				
64	(3,0)(3,1)(3,2)(3,3)(3,4)(3,5)(3,6)(3,7)				
65	(4,0)(4,1)(4,2)(4,3)(4,4)(4,5)(4,6)(4,7)	(-7,-8)			

Table B.1: Input to 4x4 Arrays and 4x4 Array Output

Clock Cycle	Search Position Output ((Y,X))							
	Blocksize(Subblock-phase1,Subblock-phase2)							
	4x8(1,5)	4x8(2,6)	4x8(3,7)	4x8(4,8)	8x4(1,5)	8x4(1,6)	8x4(1,7)	8x4(1,8)
2								
3								
4								
5	(-8,-8)							
6	(-8,-7)							
7	(-8,-6)							
8	(-8,-5)							
9	(-8,-4)	(-8,-8)			(-8,-8)	(-8,-8)		
10	(-8,-3)	(-8,-7)			(-8,-7)	(-8,-7)		
11	(-8,-2)	(-8,-6)			(-8,-6)	(-8,-6)		
12	(-8,-1)	(-8,-5)			(-8,-5)	(-8,-5)		
13	(-8,-0)	(-8,-4)	(-8,-8)		(-8,-4)	(-8,-4)		
14	(-8,1)	(-8,-3)	(-8,-7)		(-8,-3)	(-8,-3)		
15	(-8,2)	(-8,-2)	(-8,-6)		(-8,-2)	(-8,-2)		
16	(-8,3)	(-8,-1)	(-8,-5)		(-8,-1)	(-8,-1)		
17	(-8,4)	(-8,-0)	(-8,-4)	(-8,-8)	(-8,-0)	(-8,-0)	(-8,-8)	(-8,-8)
18	(-8,5)	(-8,1)	(-8,-3)	(-8,-7)	(-8,1)	(-8,1)	(-8,-7)	(-8,-7)
19	(-8,6)	(-8,2)	(-8,-2)	(-8,-6)	(-8,2)	(-8,2)	(-8,-6)	(-8,-6)
20	(-8,7)	(-8,3)	(-8,-1)	(-8,-5)	(-8,3)	(-8,3)	(-8,-5)	(-8,-5)
21		(-8,4)	(-8,-0)	(-8,-4)	(-8,4)	(-8,4)	(-8,-4)	(-8,-4)
22		(-8,5)	(-8,1)	(-8,-3)	(-8,5)	(-8,5)	(-8,-3)	(-8,-3)
23		(-8,6)	(-8,2)	(-8,-2)	(-8,6)	(-8,6)	(-8,-2)	(-8,-2)
24		(-8,7)	(-8,3)	(-8,-1)	(-8,7)	(-8,7)	(-8,-1)	(-8,-1)
25			(-8,4)	(-8,-0)			(-8,-0)	(-8,-0)
26			(-8,5)	(-8,1)			(-8,1)	(-8,1)
27			(-8,6)	(-8,2)			(-8,2)	(-8,2)
28			(-8,7)	(-8,3)			(-8,3)	(-8,3)
29			(-8,-8)	(-8,4)			(-8,4)	(-8,4)
30				(-8,5)			(-8,5)	(-8,5)
Start of Second Pass								
31				(-8,6)			(-8,6)	(-8,6)
32				(-8,7)			(-8,7)	(-8,7)
33								
34	(-8,-8)							
60				(-8,4)			(-8,4)	(-8,4)
61				(-8,5)			(-8,5)	(-8,5)
Start of second search position row/column								
62				(-8,6)			(-8,6)	(-8,6)
63				(-8,7)			(-8,7)	(-8,7)
64								
65								
66	(-7,-8)							

Table B.2: 4x8 and 8x4 outputs from adder tree

Clock Cycle	Search Position Output ((Y,X))					
	Blocksize(Subblock-phase1,Subblock-phase2)					
	8x8(1,3)	8x8(2,4)	16x8(1,2)	8x16(-,1)	8x16(-,2)	16x16(-,1)
9						
10	(-8,-8)					
11	(-8,-7)					
12	(-8,-6)					
13	(-8,-5)					
14	(-8,-4)					
15	(-8,-3)					
16	(-8,-2)					
17	(-8,-1)					
18	(-8,-0)	(-8,-8)				
19	(-8,1)	(-8,-7)	(-8,-8)			
20	(-8,2)	(-8,-6)	(-8,-7)			
21	(-8,3)	(-8,-5)	(-8,-6)			
22	(-8,4)	(-8,-4)	(-8,-5)			
23	(-8,5)	(-8,-3)	(-8,-4)			
24	(-8,6)	(-8,-2)	(-8,-3)			
25	(-8,7)	(-8,-1)	(-8,-2)			
26		(-8,-0)	(-8,-1)			
27		(-8,1)	(-8,-0)			
28		(-8,2)	(-8,1)			
29		(-8,3)	(-8,2)			
30		(-8,4)	(-8,3)			
Start of Second Pass						
31		(-8,5)	(-8,4)			
32		(-8,6)	(-8,5)			
33		(-8,7)	(-8,6)			
34			(-8,7)			
35						
51	(-8,3)	(-8,-5)	(-8,-6)	(-8,2)	(-8,-6)	(-8,-7)
52	(-8,4)	(-8,-4)	(-8,-5)	(-8,3)	(-8,-5)	(-8,-6)
53	(-8,5)	(-8,-3)	(-8,-4)	(-8,4)	(-8,-4)	(-8,-5)
54	(-8,6)	(-8,-2)	(-8,-3)	(-8,5)	(-8,-3)	(-8,-4)
55	(-8,7)	(-8,-1)	(-8,-2)	(-8,6)	(-8,-2)	(-8,-3)
56		(-8,0)	(-8,-1)	(-8,7)	(-8,-1)	(-8,-2)
57		(-8,1)	(-8,0)		(-8,0)	(-8,-1)
58		(-8,2)	(-8,1)		(-8,1)	(-8,0)
59		(-8,3)	(-8,2)		(-8,2)	(-8,1)
60		(-8,4)	(-8,3)		(-8,3)	(-8,2)
61		(-8,5)	(-8,4)		(-8,4)	(-8,3)
Start of second search position row/column						
62		(-8,6)	(-8,5)		(-8,5)	(-8,4)
63		(-8,7)	(-8,6)		(-8,6)	(-8,5)
64			(-8,7)		(-8,7)	(-8,6)
65						(-8,7)
End of first search position row/column						

Table B.3: 8x8,16x8,8x16 and 16x16 outputs from adder tree

Appendix C

Video Sequences Used

This appendix provides details on the video sequences used in this thesis. As the power consumed in an FPGA is dependent on its inputs, the video sequences used in chapters 5, 6 and 7, will influence the power consumption results given. With the power estimation methodology used, it is infeasible to use a large set of sequences. The time required would be excessive.

The estimation problem is most acute in the analysis performed in chapter 5 because, in this case, an entire in encoder is being simulated. The sequences used in chapter 5 are listed in table C.1, with a URL at which the sequences can be accessed and viewed. All are standard test sequences used in video compression research. While the test set used in chapter 5 is limited, it does contain sequences with fast motion such as *football* and sequences with a variety of different resolutions. It is also worth noting that the motion estimation and mode decision algorithms used in the encoder studied in chapter 5 are not sequence dependent. The same number of calculations are performed regardless of each sequences characteristics. Therefore, the results given in chapter 5 are less sequence dependent that what might initially be expected. However, it would have been beneficial if

Size	Sequence	URL
SIF	Football	http://www.cipr.rpi.edu/resource/sequences/sif.html
	Stefan	http://trace.eas.asu.edu/yuv/index.html
CIF	Foreman	http://www.cipr.rpi.edu/resource/sequences/sif.html
	Mobile	http://index.apple.com/~singer/sequences/mobile.mov
D1	Garden	http://www.cipr.rpi.edu/resource/sequences/sif.html

Table C.1: Test sequences used in chapter 5

Size	Sequence	URL
QCIF	Suzie	http://trace.eas.asu.edu/yuv/index.html
	Miss America	http://trace.eas.asu.edu/yuv/index.html
	Table	http://www.cipr.rpi.edu/resource/sequences/sif.html
SIF	Football	http://www.cipr.rpi.edu/resource/sequences/sif.html
CIF	Hall Monitor	http://trace.eas.asu.edu/yuv/index.html
	Paris	http://index.apple.com/~singer/sequences/paris.mov
	Mobile	http://index.apple.com/~singer/sequences/mobile.mov
VGA	Office	*Not Available*
	Riverraft	*Not Available*
	Outdoor	*Not Available*

Table C.2: Test sequences used in chapters 6 and 7

a larger sequence set could have been used.

The sequence set used in chapters 6 and 7 is shown in table C.2. The majority of sequences used are standard test sequences, for which a URL has been provided. Three sequences, *riverraft office* and *outdoor* are not. For these sequences example frames are provided in figures C.1, C.2 and C.3. The sequence set used in this case was chosen to allow the performance of the adaptive propagation algorithm to be judged. As such the sequences used contain a variety of different spatial correlations. For example, the *riverraft* sequence is predominantly horizontally corelated, where as the *office* sequence contains a mixture of vertical and horizontal correlations.

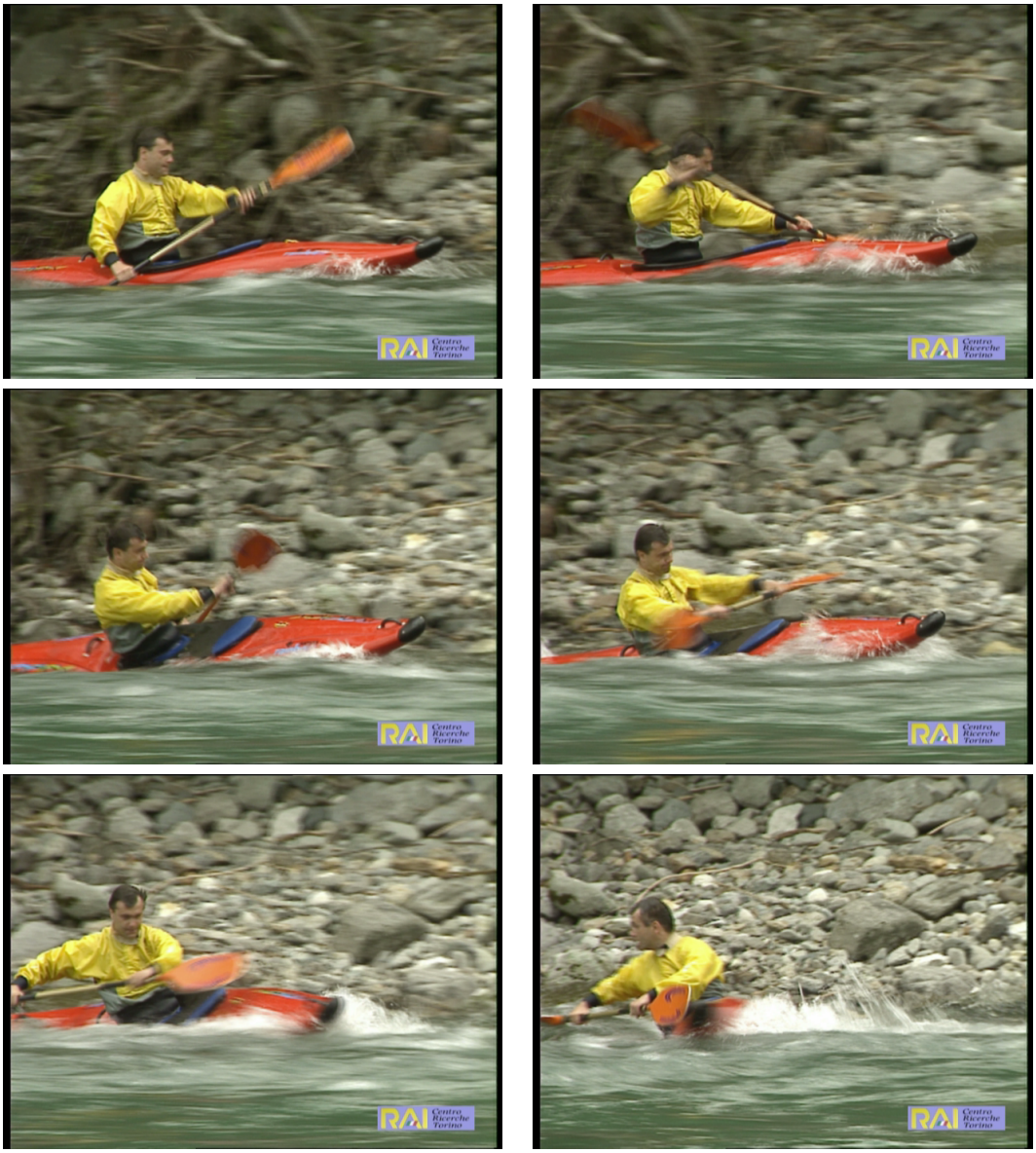

Figure C.1: Frames (from top left) 0,10,20,30,40 and 50 of riverraft test sequence

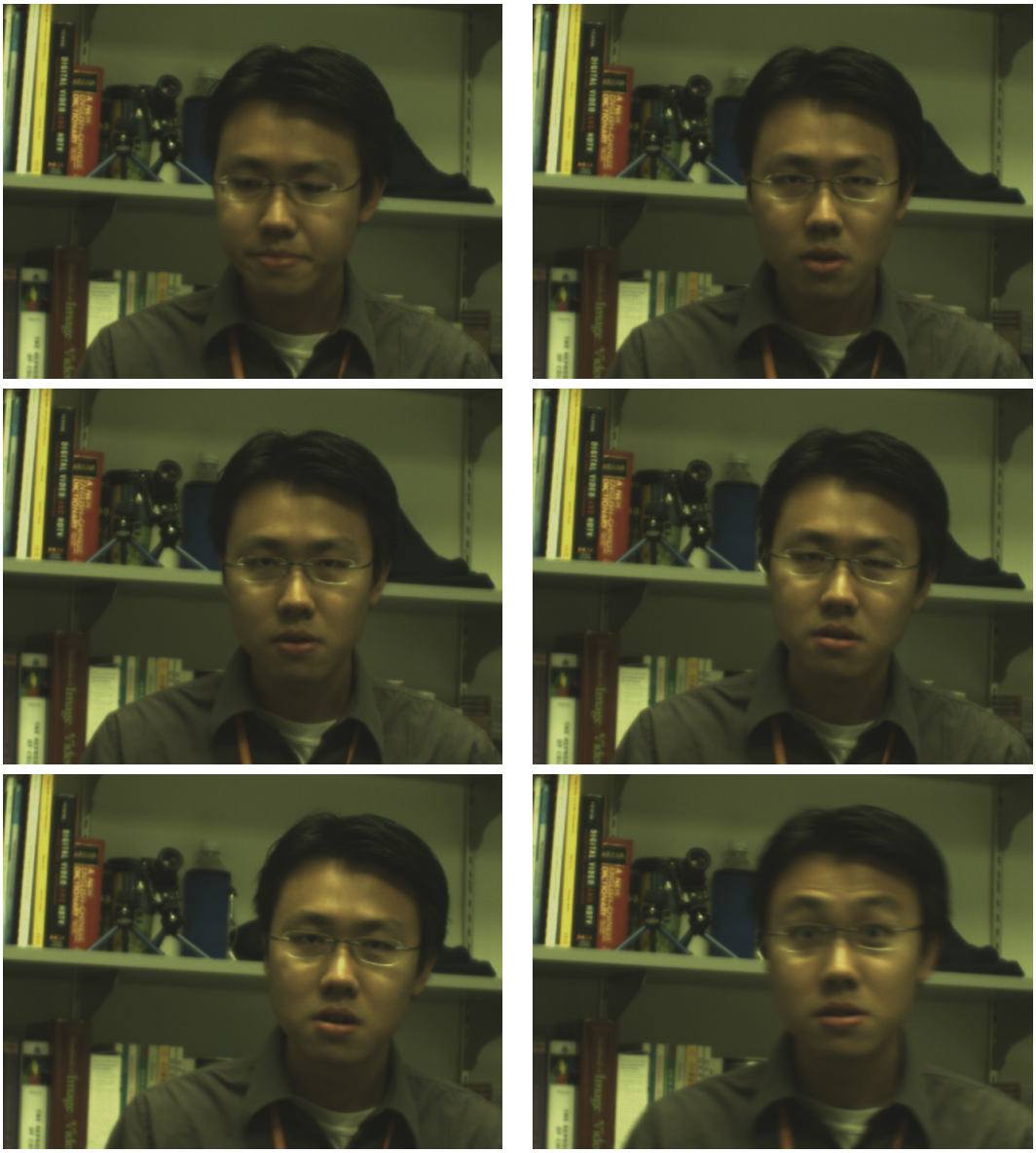

Figure C.2: Frames (from top left) 0,10,20,30,40 and 50 of office test sequence

Figure C.3: Frames (from top left) 0,10,20,30,40 and 50 of outdoor test sequence

Appendix D

H.264 Encoder Power Analysis Results

D.1 Power By Encoder Function

Tables D.1 D.2 and D.3 show the power per encoder function for a quantisation parameters of 6, 20 and 30 respectively.

	Sequence				
Encoder Function	Football	Stefan	Mobile	Foreman	Garden
Full pixel motion estimation	19.78	19.02	24.14	22.84	76.04
Fractional pixel motion estimation	10.47	10.02	12.68	11.53	39.23
Infrastructure rams	7.93	7.8	9.48	9.19	30.45
Intra Prediction	5.62	5.47	6.68	6.36	21.37
Loop filter	5.06	4.89	6.02	5.31	18.74
Forward transform and quantisation	3.47	3.4	4.08	3.97	13.11
Variable length encoding	3.01	2.68	3.39	2.82	9.58
Inverse transform and quantisation	2.22	2.09	2.54	2.35	7.78
Intra/Inter Mode decision	0.99	0.98	1.17	1.16	3.81
Reconstruction	0.7	0.64	0.82	0.58	2.14
Control Components	0.67	0.67	0.79	0.79	2.59

Table D.1: Encoder dynamic power consumption (in mW) per encoder function for a quantisation parameter of 6. Power results are in milli-watts

	Sequence				
Encoder Function	Football	Stefan	Mobile	Foreman	Garden
Full pixel motion estimation	19.84	18.76	24.08	24.99	76.16
Fractional pixel motion estimation	10.41	9.73	12.57	11.23	38.8
Infrastructure rams	7.74	7.58	9.28	8.85	29.59
Intra Prediction	5.55	5.36	6.62	6.22	21.16
Loop filter	4.9	4.75	6	5.1	18.48
Forward transform and quantisation	3.4	3.33	4.02	3.9	12.87
Variable length encoding	1.91	1.77	2.26	1.78	6.51
Inverse transform and quantisation	1.67	1.61	1.97	1.76	6.05
Intra/Inter Mode decision	0.99	0.98	1.17	1.16	3.81
Reconstruction	0.67	0.67	0.79	0.79	2.58
Control Components	0.51	0.46	0.63	0.33	1.57

Table D.2: Encoder dynamic power consumption (in mW) per encoder function for a quantisation parameter of 20. Power results are in milli-watts

	Sequence				
Encoder Function	Football	Stefan	Mobile	Foreman	Garden
Full pixel motion estimation	19.46	18.74	24.07	22.56	75.37
Fractional pixel motion estimation	9.95	9.65	12.46	10.96	38.1
Infrastructure rams	7.55	7.44	9.07	8.62	29.01
Intra Prediction	5.38	5.31	6.55	6.11	20.79
Loop filter	4.54	4.62	5.89	4.77	17.84
Forward transform and quantisation	3.35	3.31	4	3.9	12.83
Variable length encoding	1.47	1.47	1.73	1.66	5.56
Inverse transform and quantisation	1.47	1.43	1.73	1.49	5.36
Intra/Inter Mode decision	0.99	0.98	1.17	1.16	3.81
Reconstruction	0.67	0.67	0.79	0.79	2.58
Control Components	0.29	0.33	0.42	0.18	1.04

Table D.3: Encoder dynamic power consumption (in mW) per encoder function for a quantisation parameter of 30. Power results are in milli-watts

Sequence	Quant	Search Memory Load	Input Frame Load	Output Frame Save	Loop Filter Save	Loop Filter Load	Bit-stream Save	Output Frame Load
Football	6	11.18	1.8	1.21	1.45	1.47	0.65	0.07
Football	20	10.72	1.81	1.17	1.4	1.38	0.61	0.07
Football	30	9.84	1.82	1.07	1.29	1.27	0.56	0.06
Stefan	6	10.18	1.63	1.09	1.3	1.33	0.59	0.07
Stefan	20	9.69	1.65	1.04	1.24	1.24	0.55	0.06
Stefan	30	9.34	1.66	0.99	1.2	1.19	0.53	0.06
Foreman	6	12.79	2	1.36	1.62	1.64	0.73	0.08
Foreman	20	12.21	2.02	1.29	1.54	1.54	0.68	0.08
Foreman	30	11.45	2.03	1.2	1.45	1.45	0.64	0.07
Mobile	6	13.49	2.13	1.43	1.72	1.75	0.78	0.09
Mobile	20	13.21	2.13	1.41	1.69	1.68	0.75	0.09
Mobile	30	12.98	2.45	1.55	1.87	1.01	0.45	0.05
Garden	6	44.92	6.82	4.69	5.65	5.73	2.54	0.29
Garden	20	43.54	6.86	4.56	5.5	5.49	2.43	0.28
Garden	30	41.46	6.89	4.3	5.23	5.22	2.32	0.26

Table D.4: IO power consumption (in mW) attributable to various encoder memory operations

D.2 IO/SDRAM Power Consumption Results

D.2.1 IO Power Consumption

Table D.4 and Table D.5 shows the IO and SDRAM power consumption attributable to each encoder function.

Sequence	Quant	Search Memory Load	Input Frame Load	Output Frame Save	Loop Filter Save	Loop Filter Load	Bit-stream Save	Output Frame Load
Football	6	53.82	8.67	5.83	6.99	7.07	3.13	0.36
Football	20	54.85	9.27	5.97	7.16	7.07	3.13	0.36
Football	30	54.72	10.13	5.96	7.15	7.07	3.13	0.36
Stefan	6	54.21	8.68	5.8	6.94	7.07	3.13	0.36
Stefan	20	55.06	9.38	5.9	7.07	7.07	3.13	0.36
Stefan	30	55.25	9.82	5.88	7.11	7.07	3.13	0.36
Foreman	6	62.82	9.82	6.66	7.97	8.05	3.57	0.41
Foreman	20	63.67	10.5	6.73	8.02	8.05	3.57	0.41
Foreman	30	63.36	11.24	6.65	8.05	8.05	3.57	0.41
Mobile	6	61.97	9.77	6.57	7.91	8.05	3.57	0.41
Mobile	20	63.13	10.2	6.74	8.1	8.05	3.57	0.41
Mobile	30	63.82	10.52	6.78	8.23	8.05	3.57	0.41
Garden	6	201.5	30.6	21.02	25.35	25.71	11.4	1.3
Garden	20	203.95	32.11	21.34	25.78	25.71	11.4	1.3
Garden	30	204.02	33.91	21.16	25.74	25.71	11.4	1.3

Table D.5: SDRAM power consumption (in mW) attributable the various encoder memory operations

Appendix E

Fractional Estimation Power Results

This chapter details the full results used to generate Figures 7.26 and 7.27. Table E.1 gives details of the half pixel interpolator power consumption results. Table E.2 gives details of the quarter pixel interpolator and estimator power consumption results. Table E.3 gives details of the half pixel estimator power consumption results. Table E.4 gives details of the power consumption of the half and quarter pixel control functions. Table E.5 gives details of the power consumption of the quarter pixel memory.

Sequence	Horizontal Propagation	Adaptive Propagation	% reduction
Suzie	1.57	1.44	8.28
Miss america	0.91	0.86	5.49
Table	1.31	1.27	3.05
Football	6.62	6.64	-0.3
Hall monitor	5.05	4.43	12.28
Paris	6.61	5.98	9.53
Mobile	7.86	7.65	2.67
Office	11.54	10.17	11.87
Riverraft	17.49	17.45	0.23
Outdoor	12.29	12.1	1.55

Table E.1: Half pixel interpolator power consumption (mW) when adaptive propagation is and is not used

Sequence	Horizontal Propagation	Adaptive Propagation	% reduction
Suzie	1.24	1.15	7.26
Miss america	0.87	0.84	3.45
Table	1.18	1.16	1.69
Football	5.19	5.2	-0.19
Hall monitor	4.26	3.92	7.98
Paris	5.2	4.84	6.92
Mobile	6.02	5.87	2.49
Office	10.09	9.29	7.93
Riverraft	14.85	14.84	0.07
Outdoor	10.82	10.63	1.76

Table E.2: Quarter pixel estimator and interpolator power consumption (mW) when adaptive propagation is and is not used

Sequence	Horizontal Propagation	Adaptive Propagation	% reduction
Suzie	0.57	0.54	5.26
Miss america	0.43	0.42	2.33
Table	0.61	0.6	1.64
Football	2.18	2.18	0
Hall monitor	2.13	2	6.1
Paris	2.39	2.26	5.44
Mobile	2.62	2.55	2.67
Office	5.03	4.7	6.56
Riverraft	6.14	6.1	0.65
Outdoor	5.57	5.47	1.8

Table E.3: Half pixel estimator power consumption (mW) when adaptive propagation is and is not used

Sequence	Horizontal Propagation	Adaptive Propagation	% reduction
Suzie	0.38	0.38	0
Miss america	0.37	0.37	0
Table	0.38	0.38	0
Football	1.32	1.3	1.52
Hall monitor	1.37	1.37	0
Paris	1.39	1.38	0.72
Mobile	1.42	1.42	0
Office	4.31	4.31	0
Riverraft	4.51	4.52	-0.22
Outdoor	4.3	4.3	0

Table E.4: Control power consumption (mW) when adaptive propagation is and is not used

Sequence	Horizontal Propagation	Adaptive Propagation	% reduction
Suzie	0.31	0.31	0
Miss america	0.26	0.26	0
Table	0.29	0.29	0
Football	1.14	1.13	0.88
Hall monitor	1.04	0.99	4.81
Paris	1.14	1.1	3.51
Mobile	1.18	1.17	0.85
Office	3.11	3.04	2.25
Riverraft	3.66	3.66	0
Outdoor	3.13	3.12	0.32

Table E.5: Power consumpton (mW) of the quarter pixel ram when adaptive propagation is and is not used

References

[1] "Itu-t recommendation h.264 : Advanced video coding for generic audiovisual services," International Telecommunications Union, November 2007. [Online]. Available: http://www.itu.int/rec/T-REC-H.264-200711-I/en

[2] "International technology roadmap for semiconductors 2007 edition," ITRS, 2007. [Online]. Available: http://www.itrs.net/Links/2007ITRS/Home2007.htm

[3] C. Peng and T. Tran, "Hd video encoding with dsp and fpga partitioning," Texas Instruments, Inc, 2007. [Online]. Available: http://focus.ti.com.cn/cn/lit/wp/spry103/spry103.pdf

[4] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. J. Daly, "A programmable 512 gops stream processor for signal, image, and video processing," in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, 2007, pp. 272–602. [Online]. Available: http://dx.doi.org/10.1109/ISSCC.2007.373399

[5] "Tms320dm6467 digital media system-on-chip datasheet," Texas Instruments, Inc, February 2009. [Online]. Available: http://focus.ti.com/docs/prod/folders/print/tms320dm6467.html

[6] S. Hu, Z. Zhang, M. Zhang, and T. Sheng, "Optimization of memory allocation for h.264 video decoder on digital signal processors," in *Image and Signal*

Processing, 2008. CISP '08. Congress on, vol. 2, 2008, pp. 71–75. [Online]. Available: http://dx.doi.org/10.1109/CISP.2008.173

[7] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors," *Computer*, vol. 36, no. 8, pp. 54–62, 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1220582

[8] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 688–703, 2003. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2003.815168

[9] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with h.264/avc: tools, performance, and complexity," *Circuits and Systems Magazine, IEEE*, vol. 4, no. 1, pp. 7–28, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1286980

[10] M. J. Garrido, C. Sanz, M. Jimenez, and J. M. Menesses, "An fpga implementation of a flexible architecture for h.263 video coding," *Consumer Electronics, IEEE Transactions on*, vol. 48, no. 4, pp. 1056–1066, 2002. [Online]. Available: http://dx.doi.org/10.1109/TCE.2003.1196439

[11] K. Denolf, A. Chirila-Rus, R. Turney, P. Schumacher, and K. Vissers, "Memory efficient design of an mpeg-4 video encoder for fpgas," in *Field Programmable Logic and Applications, 2005. International Conference on*, 2005, pp. 391–396. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1515753

[12] O. Lehtoranta, E. Salminen, A. Kulmala, M. Hannikainen, and T. D. Hamalainen, "A parallel mpeg-4 encoder for fpga based multiprocessor soc," in *Field*

Programmable Logic and Applications, 2005. International Conference on, 2005, pp. 380–385. [Online]. Available: http://dx.doi.org/10.1109/FPL.2005.1515751

[13] V. Liguori and K. Wong, "Designing a real time hdtv 1080p baseline h.264/avc encoder core," in *DesignCon*. International Engineering Consortium, February 2006. [Online]. Available: http://www.cast-inc.com/info/events/shows/dcon06/1-WP1--Vincenzo%20Liguori.pdf

[14] K. Babionitakis, G. Doumenis, G. Georgakarakos, G. Lentaris, K. Nakos, D. Reisis, I. Sifnaios, and N. Vlassopoulos, "A real-time h.264/avc vlsi encoder architecture," *Journal of Real-Time Image Processing*, March 2008. [Online]. Available: http://dx.doi.org/10.1007/s11554-007-0054-9

[15] P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, "Run-time integration of reconfigurable video processing systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 1003–1016, 2007. [Online]. Available: http://dx.doi.org/10.1109/TVLSI.2007.902203

[16] Y. Shin, S.-I. Chae, and K. Choi, "Partial bus-invert coding for power optimization of application-specific systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 2, pp. 377–383, 2001. [Online]. Available: http://dx.doi.org/10.1109/92.924059

[17] T.-C. Chen, Y.-W. Huang, and L.-G. Chen, "Fully utilized and reusable architecture for fractional motion estimation of h.264/avc," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, vol. 5, 2004, pp. V–9–12 vol.5. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1327034

[18] A. M. Campos, F. B. Merelo, M. M. Peiro, and J. C. Esteve, "High parallel-pipeline integer-pel and fractional-pel motion estimation vlsi architectures for

h.264/avc," in *VLSI Circuits and Systems III*, Valente, K. Eshraghian, and F. B. Tobajas, Eds., vol. 6590, no. 1. SPIE, 2007. [Online]. Available: http://dx.doi.org/10.1117/12.724042

[19] S. Oktem and I. Hamzaoglu, "An efficient hardware architecture for quarter-pixel accurate h.264 motion estimation," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, 2007, pp. 444–447. [Online]. Available: http://dx.doi.org/10.1109/DSD.2007.4341507

[20] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression.* Wiley, 2003.

[21] G. J. Sullivan and T. Wiegand, "Video compression - from concepts to the h.264/avc standard," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, 2005. [Online]. Available: http://dx.doi.org/10.1109/JPROC.2004.839617

[22] R. Schafer and T. Sikora, "Digital video coding standards and their role in video communications," *Proceedings of the IEEE*, vol. 83, no. 6, pp. 907–924, 1995. [Online]. Available: http://dx.doi.org/10.1109/5.387092

[23] "H.264 baseline video encoder ip core," 4i2i Communications Ltd, May 2007. [Online]. Available: http://www.4i2i.com/downloads/H264BaseEncXilinxIP.pdf

[24] Y.-C. Chang, W.-M. Chao, and L.-G. Chen, "Platform-based mpeg-4 video encoder soc design," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, 2004, pp. 251–256. [Online]. Available: http://dx.doi.org/10.1109/SIPS.2004.1363058

[25] Ben, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, and H. Levi, "An fpga implementation of hw/sw codesign architecture for h.263 video coding," *AEU - International Journal of Electronics and Communications*, vol. 61, no. 9, pp. 605–620, October 2007. [Online]. Available: http://dx.doi.org/10.1016/j.aeue.2006.11.001

[26] M. Ikeda, T. Kondo, K. Nitta, K. Suguri, T. Yoshitome, T. Minami, H. Iwasaki, K. Ochiai, J. Naganuma, M. Endo, Y. Tashiro, H. Watanabe, N. Kobayashi, T. Okubo, and R. Kasai, "Superenc: Mpeg-2 video encoder chip," *IEEE Micro*, vol. 19, no. 4, pp. 56–65, 1999. [Online]. Available: http://dx.doi.org/10.1109/40.782568

[27] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an hdtv720p 30 frames/s h.264/avc encoder," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 6, pp. 673–688, 2006. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2006.873163

[28] T.-C. Chen, Y.-W. Huang, and L.-G. Chen, "Analysis and design of macroblock pipelining for h.264/avc vlsi architecture," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, 2004, pp. II–273–6 Vol.2. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1329261

[29] Y.-K. Chen, E. Q. Li, X. Zhou, and S. Ge, "Implementation of h.264 encoder and decoder on personal computers," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 509–532, April 2006. [Online]. Available: http://dx.doi.org/10.1016/j.jvcir.2005.05.004

[30] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Hierarchical parallelization of an h.264/avc video encoder," in *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, 2006, pp. 363–368. [Online]. Available: http://dx.doi.org/10.1109/PARELEC.2006.42

[31] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *Circuits and Systems for Video Technology,*

IEEE Transactions on, vol. 4, no. 4, pp. 438–442, 1994. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=313138

[32] M. J. Chen, L. G. Chen, and T. D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 4, no. 5, pp. 504–509, 1994. [Online]. Available: http://dx.doi.org/10.1109/76.322998

[33] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 3, pp. 313–317, 1996. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=499840

[34] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *Image Processing, IEEE Transactions on*, vol. 9, no. 2, pp. 287–290, 2000. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=821744

[35] C. Zhu, X. Lin, and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 12, no. 5, pp. 349–355, 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1003474

[36] Z.-L. He, C.-Y. Tsui, K.-K. Chan, and M. L. Liou, "Low-power vlsi design for motion estimation using adaptive pixel truncation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 10, no. 5, pp. 669–678, 2000. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=856445

[37] A. Bahari, T. Arslan, and A. T. Erdogan, "Low power variable block size motion estimation using pixel truncation," in *Circuits and Systems, 2007. ISCAS 2007.*

IEEE International Symposium on, 2007, pp. 3663–3666. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4253475

[38] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *Image Processing, IEEE Transactions on*, vol. 4, no. 1, pp. 105–107, 1995. [Online]. Available: http://dx.doi.org/10.1109/83.350809

[39] A. M. Tourapis, O. C. Au, and M. L. Liou, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 12, no. 10, pp. 934–947, 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1058224

[40] C. H. Hsieh and T. P. Lin, "Vlsi architecture for block-matching motion estimation algorithm," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 2, no. 2, pp. 169–175, 1992. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=143416

[41] D. Tzovaras and M. G. Strintzis, "Motion and disparity field estimation using rate-distortion optimization," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 8, no. 2, pp. 171–180, 1998.

[42] Y.-W. Huang, C.-Y. Chen, C.-H. Tsai, C.-F. Shen, and L.-G. Chen, "Survey on block matching motion estimation algorithms and architectures with new results," *The Journal of VLSI Signal Processing*, vol. 42, no. 3, pp. 297–320, March 2006. [Online]. Available: http://dx.doi.org/10.1007/s11265-006-4190-4

[43] C.-Y. Chen, S.-Y. Chien, Y.-W. Huang, T.-C. Chen, T.-C. Wang, and L.-G. Chen, "Analysis and architecture design of variable block-size motion estimation for h.264/avc," *Circuits and Systems I: Regular Papers, IEEE Transactions on [see also Circuits and Systems I: Fundamental Theory and Applications,*

IEEE Transactions on], vol. 53, no. 3, pp. 578–593, 2006. [Online]. Available: http://dx.doi.org/10.1109/TCSI.2005.858488

[44] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 3, no. 2, pp. 148–157, 1993. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=212720

[45] H.-W. Cheng and L.-R. Dung, "A content-based methodology for power-aware motion estimation architecture," *Circuits and Systems II: Express Briefs, IEEE Transactions on [see also Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on]*, vol. 52, no. 10, pp. 631–635, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1519649

[46] Z. Liu, Y. Song, T. Ikenaga, and S. Goto, "Low-pass filter based vlsi oriented variable block size motion estimation algorithm for h.264," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 2, 2006, p. II. [Online]. Available: http://dx.doi.org/10.1109/ICASSP.2006.1660327

[47] M.-J. Chen, L.-G. Chen, T.-D. Chiueh, and Y.-P. Lee, "A new block-matching criterion for motion estimation and its implementation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 5, no. 3, pp. 231–236, 1995. [Online]. Available: http://dx.doi.org/10.1109/76.401100

[48] H. Gharavi and M. Mills, "Blockmatching motion estimation algorithms-new results," *Circuits and Systems, IEEE Transactions on*, vol. 37, no. 5, pp. 649–651, 1990. [Online]. Available: http://dx.doi.org/10.1109/31.55010

[49] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *Image Processing, IEEE*

Transactions on, vol. 9, no. 3, pp. 501–504, 2000. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=826786

[50] Z. Chen, J. Xu, Y. He, and J. Zheng, "Fast integer-pel and fractional-pel motion estimation for h.264/avc," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 264–290, April 2006. [Online]. Available: http://dx.doi.org/10.1016/j.jvcir.2004.12.002

[51] Y.-W. Huang, S.-Y. Chien, B.-Y. Hsieh, and L.-G. Chen, "Global elimination algorithm and architecture design for fast block matching motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 14, no. 6, pp. 898–907, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1302172

[52] T.-C. Chen, Y.-H. Chen, S.-F. Tsai, and L.-G. Chen, "Architecture design of low power integer motion estimation for h. 264/avc," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 3, 2006, pp. III–900–III–903. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1660800

[53] S. Yang, W. Wolf, and N. Vijaykrishnan, "Power and performance analysis of motion estimation based on hardware and software realizations," *Computers, IEEE Transactions on*, vol. 54, no. 6, pp. 714–726, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1461359

[54] R. Tessier, V. Betz, D. Neto, A. Egier, and T. Gopalsamy, "Power-efficient ram mapping algorithms for fpga embedded memory blocks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 278–290, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4068934

[55] K. M. Yang, M. T. Sun, and L. Wu, "A family of vlsi designs for the motion compensation block-matching algorithm," *Circuits and Systems, IEEE Transactions on*, vol. 36, no. 10, pp. 1317–1325, 1989. [Online]. Available: http://dx.doi.org/10.1109/31.44348

[56] S. Y. Yap and J. V. Mccanny, "A vlsi architecture for variable block size video motion estimation," *Circuits and Systems II: Express Briefs, IEEE Transactions on [see also Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on]*, vol. 51, no. 7, pp. 384–389, 2004. [Online]. Available: http://dx.doi.org/10.1109/TCSII.2004.829555

[57] B. Li and P. Leong, "Serial and parallel fpga-based variable block size motion estimation processors," *Journal of Signal Processing Systems*, vol. 51, no. 1, pp. 77–98, April 2008. [Online]. Available: http://dx.doi.org/10.1007/s11265-007-0143-9

[58] V. L. Do and K. Y. Yun, "A low-power vlsi architecture for full-search block-matching motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 8, no. 4, pp. 393–398, 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=709406

[59] M. Jiang, D. Crookes, S. Davidson, and R. Turner, "Low-power systolic array processor architecture for fsbm video motion estimation," *Electronics Letters*, vol. 42, no. 20, pp. 1146–1147, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1706025

[60] S. Lopez, F. Tobajas, A. Villar, V. de Armas, J. F. Lopez, and R. Sarmiento, "Low cost efficient architecture for h.264 motion estimation," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 2005, pp. 412–415 Vol. 1. [Online]. Available: http://dx.doi.org/10.1109/ISCAS.2005.1464612

[61] H.-M. Jong, Liang-Gee, and T.-D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 4, no. 4, pp. 407–416, 1994. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=313135

[62] Y.-L. Xi, C.-Y. Hao, Y.-Y. Fan, and H.-Q. Hu, "A fast block-matching algorithm based on adaptive search area and its vlsi architecture for h.264/avc," *Signal Processing: Image Communication*, vol. 21, no. 8, pp. 626–646, September 2006. [Online]. Available: http://dx.doi.org/10.1016/j.image.2006.05.001

[63] S. Dutta and W. Wolf, "A flexible parallel architecture adapted to block-matching motion-estimation algorithms," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 6, no. 1, pp. 74–86, 1996. [Online]. Available: http://dx.doi.org/10.1109/76.486422

[64] K. Babionitakis, G. Doumenis, G. Georgakarakos, G. Lentaris, K. Nakos, D. Reisis, I. Sifnaios, and N. Vlassopoulos, "A real-time motion estimation fpga architecture," *Journal of Real-Time Image Processing*, vol. 3, no. 1, pp. 3–20, March 2008. [Online]. Available: http://dx.doi.org/10.1007/s11554-007-0070-9

[65] M. Ribeiro and L. Sousa, "A run-time reconfigurable processor for video motion estimation," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 726–729. [Online]. Available: http://dx.doi.org/10.1109/FPL.2007.4380755

[66] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching vlsi architecture," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 12, no. 1, pp. 61–72, 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=981846

[67] C.-Y. Chen, C.-T. Huang, Y.-H. Chen, and L.-G. Chen, "Level c+ data reuse scheme for motion estimation with corresponding coding orders," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 4, pp. 553–558, 2006. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2006.871388

[68] T. Wedi and H. G. Musmann, "Motion- and aliasing-compensated prediction for hybrid video coding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 577–586, 2003. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2003.815171

[69] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, "Performance analysis of hardware oriented algorithm modifications in h.264," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, vol. 2, 2003, pp. II–493–6 vol.2. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1202411

[70] C. A. Rahman and W. Badawy, "A quarter pel full search block motion estimation architecture for h.264/avc," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, 2005, pp. 4 pp.+. [Online]. Available: http://dx.doi.org/10.1109/ICME.2005.1521448

[71] W. Lee, S. Lee, and J. Kim, "Pipelined intra prediction using shuffled encoding order for h.264/avc," in *TENCON 2006. 2006 IEEE Region 10 Conference*, 2006, pp. 1–4. [Online]. Available: http://dx.doi.org/10.1109/TENCON.2006.343970

[72] Y. J. Liang and K. El-Maleh, "Low-complexity intra/inter mode-decision for h.264/avc video coder," in *Intelligent Multimedia, Video and Speech Processing, 2004. Proceedings of 2004 International Symposium on*, 2004, pp. 53–56. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1433998

[73] C. Kim and C. C. J. Kuo, "A feature-based approach to fast h.264 intra/inter mode decision," in *Circuits and Systems, 2005. ISCAS 2005. IEEE*

International Symposium on, 2005, pp. 308–311 Vol. 1. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1464586

[74] C. S. Kannangara, I. E. G. Richardson, M. Bystrom, J. R. Solera, Y. Zhao, A. Maclennan, and R. Cooney, "Low-complexity skip prediction for h.264 through lagrangian cost estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 2, pp. 202–208, 2006. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2005.859026

[75] C. F. Chen and K. K. Pang, "The optimal transform of motion-compensated frame difference images in a hybrid coder," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 40, no. 6, pp. 393–397, 1993.

[76] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in h.264/avc," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 598–603, 2003. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2003.814964

[77] K. Suh, S. Park, and H. Cho, "An efficient hardware architecture of intra prediction and tq/iqit module for h.264 encoder." *ETRI Journal*, vol. 27, no. 5, pp. 511–524, October 2005. [Online]. Available: http://etrij.etri.re.kr/Cyber/servlet/BrowseAbstract?paperid=S40501-0032

[78] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, "Parallel 4/spl times/4 2d transform and inverse transform architecture for mpeg-4 avc/h.264," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 2, 2003, pp. II–800–II–803 vol.2. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1206095

[79] K.-H. Chen, J.-I. Guo, and J.-S. Wang, "A high-performance direct 2-d transform coding ip design for mpeg-4avc/h.264," *Circuits and Systems for*

Video Technology, IEEE Transactions on, vol. 16, no. 4, pp. 472–483, 2006. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2006.872782

[80] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 620–636, 2003. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2003.815173

[81] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblocking filter," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 614–619, 2003. [Online]. Available: http://dx.doi.org/10.1109/TCSVT.2003.815175

[82] A. Amara, F. Amiel, and T. Ea, "Fpga vs. asic for low power applications," *Microelectronics Journal*, vol. 37, no. 8, pp. 669–677, August 2006. [Online]. Available: http://dx.doi.org/10.1016/j.mejo.2005.11.003

[83] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," in *FPGA'06: Proceedings of the internation symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2006, pp. 21–30. [Online]. Available: http://dx.doi.org/10.1145/1117201.1117205

[84] "Cyclone ii device handbook volume 1," Altera Corporation, June 2006. [Online]. Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf

[85] I. Xilinx, "Spartan-3 fpga data sheet," June 2008. [Online]. Available: http://www.xilinx.com/support/documentation/spartan-3_data_sheets.htm

[86] Altera, "Cyclone ii fpga family datasheet," February 2008. [Online]. Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii51001.pdf

[87] ——, "Cyclone iii device datasheet," October 2008. [Online]. Available: http://www.altera.com/literature/hb/cyc3/cyc3_ciii5v2.pdf

[88] J. H. Anderson and F. N. Najm, "Power estimation techniques for fpgas," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 10, pp. 1015–1027, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1336847

[89] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, "A 90nm low-power fpga for battery-powered applications," in *FPGA'06: Proceedings of the internation symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2006, pp. 3–11. [Online]. Available: http://dx.doi.org/10.1145/1117201.1117203

[90] T. Tuan and B. Lai, "Leakage power analysis of a 90nm fpga," in *Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE 2003*, 2003, pp. 57–60. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1249359

[91] F. Li, Y. Lin, L. He, and J. Cong, "Low-power fpga using pre-defined dual-vdd/dual-vt fabrics," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2004, pp. 42–50. [Online]. Available: http://dx.doi.org/10.1145/968280.968288

[92] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in fpgas using region-constrained placement," in *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2004, pp. 51–58. [Online]. Available: http://dx.doi.org/10.1145/968280.968289

[93] F. Li, Y. Lin, and L. He, "Fpga power reduction using configurable dual-vdd," in *Design Automation Conference, 2004. Proceedings. 41st*, 2004, pp. 735–740. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1322580

[94] J. H. Anderson and F. N. Najm, "Active leakage power optimization for fpgas," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 3, pp. 423–437, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1597379

[95] ——, "Power-aware technology mapping for lut-based fpgas," in *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on*, 2002, pp. 211–218. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1188684

[96] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware fpga cad algorithms," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, 2003, pp. 701–708. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1257886

[97] S. J. E. Wilton, S.-S. Ang, and W. Luk, "The impact of pipelining on energy per operation in field-programmable gate arrays," in *Lecture Notes in Computer Science : Field Programmable Logic and Application*, 2004, pp. 719–728. [Online]. Available: http://www.springerlink.com/content/3peu6h9vv32x8akk

[98] H. Lim, K. Lee, Y. Cho, and N. Chang, "Flip-flop insertion with shifted-phase clocks for fpga power reduction," in *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 335–342. [Online]. Available: http://portal.acm.org/citation.cfm?id=1129601.1129651

[99] R. Fischer, K. Buchenrieder, and U. Nageldinger, "Reducing the power consumption of fpgas through retiming," in *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the*, 2005, pp. 89–94. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1409905

[100] S. Khawam, T. Arslan, and F. Westali, "Embedded reconfigurable array targeting motion estimation applications," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 2, 2003, pp. II–760–II–763 vol.2. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1206085

[101] J. Rose, "Hard vs. soft: the central question of pre-fabricated silicon," in *Multiple-Valued Logic, 2004. Proceedings. 34th International Symposium on*, 2004, pp. 2–5. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1319911

[102] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware ram mapping for fpga embedded memory blocks," in *FPGA'06: Proceedings of the internation symposium on Field programmable gate arrays*. New York, NY, USA: ACM Press, 2006, pp. 189–198. [Online]. Available: http://dx.doi.org/10.1145/1117201.1117229

[103] "Quartus 2 version 8.0 handbook volume 2: Design implementation and optimization," Altera Corporation, May 2008. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii52016.pdf

[104] S. J. E. Wilton, "Implementing logic in fpga embedded memory arrays: architectural implications," in *Custom Integrated Circuits Conference, 1998., Proceedings of the IEEE 1998*, 1998, pp. 269–272. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=694978

[105] S. Y. L. Chin, C. S. P. Lee, and S. J. E. Wilton, "Power implications of implementing logic using fpga embedded memory arrays," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, 2006, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4100962

[106] Synplicity, "Gated clock conversion with synplicity's synthesis products," web. [Online]. Available: #

[107] Y. Zhang, J. Roivainen, and A. Mammela, "Clock-gating in fpgas: A novel and comparative evaluation," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. '584–590. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1690091

[108] "H.261/3 and mpeg-4 video encoder ip core datasheet," 4i2i Communcations, 2004.

[109] "Opb ipif (v3.01c)," Xilinx, December 2005. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/opb_ipif.pdf

[110] "Xilinx spartan-3 evaluation kit," Avnet, Inc., 2004. [Online]. Available: http://www.em.avnet.com/ctf_shared/evk/df2df2usa/Xilinx%20Spartan-3%20Evaluation%20Kit%20-%20Brief%20022504F.pdf

[111] "Audio/video module," Avnet, Inc., 2004. [Online]. Available: http://www.em.avnet.com/ctf_shared/evk/df2df2usa/Audio_Video_Module%_Brief_040904F.pdf

[112] "Communications/memory module," Avnet, Inc., 2002. [Online]. Available: http://www.em.avnet.com/ctf_shared/evk/df2df2usa/Communications_Memory_Module_Brief_021003F.pdf

[113] "Opb synchronous dram (sdram) controller (v1.00e)," Xilinx, July 2005. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/opb_sdram.pdf

[114] "Opb ethernet media access controller (emac) (v1.04a)," Xilinx, November 2005. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/opb_ethernet.pdf

[115] C. Zhu, "Rfc 2190: Rtp payload format for h.263 video streams," Internet Engineering Task Force, September 1997. [Online]. Available: http://www.faqs.org/rfcs/rfc2190.html

[116] C. Bormann, L. Cline, T. Gardos, C. Maciocco, D. Newell, J. Ott, G. Sullivan, S. Wenger, and C. Zhu, "Rfc 2329: Rtp payload format for the 1998 version of itu-t rec. h.263 video (h.263+)," Internet Engineering Task Force, October 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2429.txt

[117] 4i2i, "H.264 video encoder ip core datasheet," August 2005.

[118] "The efficiency of the ddr & ddr2 sdram controller compiler," Altera Corporation, December 2004. [Online]. Available: http://www.altera.com/literature/wp/wp_ddr_sdram_efficiency.pdf

[119] Micron, "Micron sdram mt48lc2m32b2 datasheet," March 2007. [Online]. Available: http://download.micron.com/pdf/datasheets/dram/sdram/64MSDRAMx32.pdf

[120] *Modelsim SE Reference Manual 6.3f*, Mentor Graphics Corporation, March 2008.

[121] "Quartus ii version 8.0 handbook volume 3: Verification," Altera Corporation, May 2008. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii5v3.pdf

[122] "Micron system power calculator," Micron Technology, Inc, April 2001. [Online]. Available: http://www.micron.com/support/part_info/powercalc

[123] "Stratix ii and virtex-4 power comparison," Altera Corporation, August 2005. [Online]. Available: http://www.altera.com/products/devices/stratix-fpgas/stratix-ii/stratix-ii/features/st2-competitive.html?GSA_pos=3\&WT.oss_r=1\&WT.oss=powerplay%20accuracy

[124] P. E. Landman and J. M. Rabaey, "Architectural power analysis: The dual bit type method," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 173–187, 1995. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=386219

[125] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power i/o," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, no. 1, pp. 49–58, 1995. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=365453

[126] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "A coding framework for low-power address and data busses," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 7, no. 2, pp. 212–221, 1999. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=766748

[127] K. Mohanram and S. Rixner, "Context-independent codes for off-chip interconnects," in *Power-Aware Computer Systems*, 2005, pp. 107–119. [Online]. Available: http://dx.doi.org/10.1007/11574859_8

[128] A. Bahari, T. Arslan, and A. T. Erdogan, "Interframe bus encoding technique for low power video compression," in *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, 2007, pp. 691–698. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4092122

[129] C. Kretzschmar, R. Siegmund, and D. Müller, "Low power encoding techniques for dynamically reconfigurable hardware," *The Journal of Supercomputing*, vol. 26, no. 2, pp. 185–203, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1024451718410

[130] "H.264/avc jm reference software," Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, August 2008. [Online]. Available: http://iphome.hhi.de/suehring/tml/

[131] B. Balasubramaniyam, "Stereoscopic video coding," Ph.D. dissertation, Loughborough University, September 2006.

[132] A. Woods, "The application of stereoscopic video to underwater remotely operated vehicles." *APPEA Journal*, vol. 37, pp. 797–800, 1997.

[133] David, "Skill acquisition and task performance in teleoperation using monoscopic and stereoscopic video remote viewing," in *Proceedings of the Human Factors Society*, 1991, pp. 1367–1371.

[134] M. Lukacs, "Predictive coding of multi-viewpoint image sets," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on.*, vol. 11, 1986, pp. 521–524.

[135] Mel, Priyan, S. Sethuraman, and Angel, "Compression of stereo image pairs and streams," in *Stereoscopic Displays and Virtual Reality Systems.Proceedings of SPIE*, vol. 2177, 1994.

[136] S. Sun and S. Lei, "Stereo-view video coding using h.264 tools," in *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 5685, 2005, pp. 177–184.

[137] K. Kamikura, H. Watanabe, H. Jozawa, H. Kotera, and S. Ichinose, "Global brightness-variation compensation for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 8, 1998.

[138] I. Dinstein, G. Guy, J. Rabany, J. Tzelgov, and A. Henik, "On stereo image coding," in *Proceedings - International Conference on Pattern Recognition*, 1988, pp. 357–359.

[139] A. Puri, R. V. Kollarits, and B. G. Haskell, "Basics of stereoscopic video, new compression results with mpeg-2 and a proposal for mpeg-4," *Signal Processing: Image Communication*, vol. 10, no. 1-3, pp. 201–234, 1997.

[140] W. Yang, K. Ngan, J. Lim, and K. Sohn, "Joint motion and disparity fields estimation for stereoscopic video sequences," *Signal Processing: Image Communication*, vol. 20, no. 3, pp. 265–276, 2005.

[141] A. Mancini, "Disparity estimation and intermediate view reconstruction for novel applications in stereoscopic video," Ph.D. dissertation, McGill University, 1998.

[142] J. D. Oh and C. C. J. Kuo, "A stereo video coding scheme based on h.264," in *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 5909, 2005, pp. 1–10.